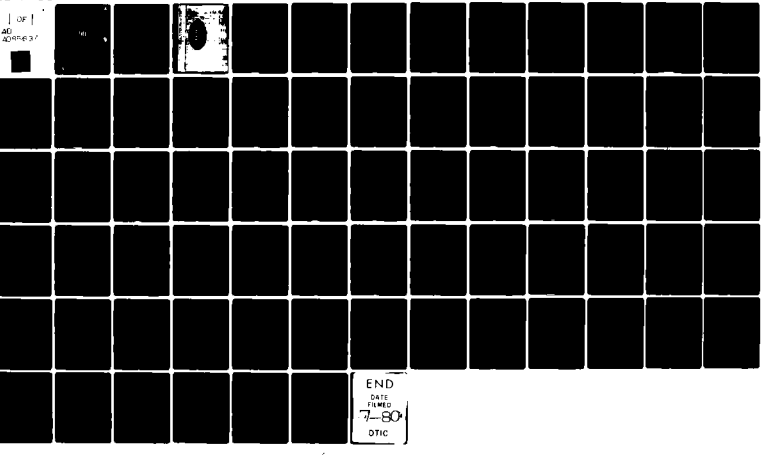
AD-A085 637 ARMY SATELLITE COMMUNICATIONS AGENCY FORT MONMOUTH NJ F/G 9/2
PROJECT ARIES. Z80 CROSS ASSEMBLER AND LINKER USER'S MANUAL.(U)
APR 80 R L CONN

UNCLASSIFIED NL



END
DATE FILMED
7-80
DTIC

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO. AD-A085637
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
PROJECT ARIES
Z80 CROSS ASSEMBLER and LINKER USER'S MANUAL

5. TYPE OF REPORT & PERIOD COVERED
Final
5/79 to 5/80

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Richard L. Conn

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
US Army Satellite Communications Agency
USA CORADCOM, Attn: DRCPM-SC-4G
Ft Monmouth, NJ 07703

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Elt: 6.11.01.A
Proj: 1L1 61101 A91A
Task: 33 Work Unit: 131

11. CONTROLLING OFFICE NAME AND ADDRESS
US Army Satellite Communications Agency
USA CORADCOM, Attn: DRCPM-SC-4G
Ft Monmouth, NJ 07703

12. REPORT DATE
April, 1980

13. NUMBER OF PAGES
65

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

LEVEL II

16. DISTRIBUTION STATEMENT (of this Report)
Distribution Unlimited

DTIC ELECTE JUN 16 1980 D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Distribution Unlimited

18. SUPPLEMENTARY NOTES
Source Listings to Z80 Cross Assembler and Linking Loader included as part of report.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Z80 microprocessor, assembler, linker, loader, PASCAL

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The Z80 Cross Assembler described by this document is a Zilog-standard cross assembler written in PASCAL. It supports all the Zilog-standard mnemonics, but its pseudo-ops are unique to this assembler(to some extent). This assembler produces relocatable code which may be later linked and loaded by its companion linking loader.

This cross assembler was customized for application with the ARIAN II operating system. It is designed to be run on a CDC 6000 or CYBER class

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A085637

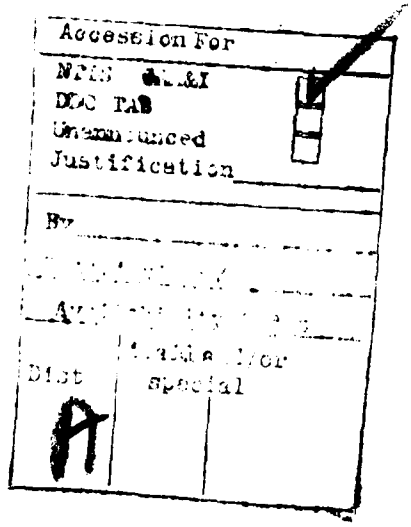
DDC FILE COPY

80 6 12 024

20. (con't)

machine. The intended operating environment is in a timesharing mode on the host computer with a microcomputer linked to the host for communications and, specifically, downloading.

- End of Information -



Prod. # Aries.

Z80 CROSS ASSEMBLER and LINKER USER'S MANUAL

by

RICHARD L. CONN

11 Aug 49

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

(16) 1L161101A71A (17) 33

Z80 Cross Assembler and Linker
Originally Written, Maintained, and Supported
on University of Illinois' CDC CYBER-175 Computer System

by

GEORGE LEHMANN, JR.

May, 1978

Modified, Maintained, and Supported
on Picatinny Arsenal's CDC 6500/6600 Computer Systems

by

RICHARD L CONN

April, 1980

Automated Systems Division, Programs Directorate
US Army Satellite Communications Agency
Fort Monmouth, NJ 07703

(9) Final rpt. May 79 - May 80

041

TABLE of CONTENTS

Z80ASM - Z80 CROSS ASSEMBLER and LINKER USER'S MANUAL

CHAPTER 1

Introduction

The Z80 Cross Assembler was written for a number of reasons. Among them are: (1) the need of an assembler customized for the Project ARIES applications, (2) the inefficiency of the MAC80 cross assembler, and (3) the lack of linking loader facilities for Z80 cross software. This assembler provides the basis for a linking loader environment by establishing a relocatable object format.

This assembler is designed to fully support the Zilog-standard mnemonics for the Z80 op codes. All Zilog-standard mnemonics are implemented by the assembler. The pseudo-ops, however, are mainly unique to this assembler.

The Z80 Cross Assembler, hereafter referred to as ASMZ80, produces three files as output. The first file is the output file to the user's terminal or batch job stream. Messages are sent to this file by the assembler. The second file is the assembler listing file. It is in a paged format suitable for line printer listing. Finally, the third file is the object file. It is in the Project ARIES object format.

CHAPTER 2

Use of the Assembler

The assembler is very easy to use if one is familiar with timesharing on the CDC 6500/6600 under INTERCOM. If not, see ARRADCOM MISD consultants at Picatinny Arsenal on how to get started. Basically, one will need to put his Z80 program into a text file by using an editor. When this has been done, the user may then save his program as a local file and use the assembler. Two local files are generated by the assembler, a program listing file and an object file. This object file is called a paper-tape object file because it is suitable for punching on paper tape. The paper tape format also makes the file suitable for transmission over telephone lines to a microcomputer.

2.1 Pseudo-Ops

The pseudo-ops are one of the features that make one assembler different from another. For this reason, the pseudo-ops of ASMZ80 are listed in some detail.

DB - define byte. The operand(s) of this pseudo-op are evaluated and emitted one byte per operand. Strings (enclosed in single quotes) are an exception to this, with one byte being emitted for each character in the string (not counting the beginning and ending single quotes).

DEFB -- define byte. This pseudo-op was included to help maintain compatability with the Zilog-standard pseudo-ops. The DEFB function is a subfunction of DB, and DEFB in Z80ASM is implemented exactly as DB.

DEFM -- define message. Same as DEFB.

DEFS -- define storage. This pseudo-op was included to help maintain compatability with the Zilog-standard pseudo-ops. The DEFS function is a subfunction of DS, and DEFS in Z80ASM is implemented exactly as DS.

DEFW -- define word. This pseudo-op was included to help maintain compatability with the Zilog-standard pseudo-ops. The DEFW function is a subfunction of DW, and DEFW in Z80ASM is implemented exactly as DW.

DISPLAY - display the value of an expression on the user's terminal. This is useful when the user wishes to know a particular value computed by the assembler without having to scan through the listing. For example, a typical use of DISPLAY would be to display the range of an assembly.

DS - define storage. The operand field is evaluated, and that many bytes of memory are reserved starting with the current value of the memory location counter.

DW - define word. The operand(s) of this pseudo-op are evaluated and emitted one word (two bytes) per operand.

END - end of program. This pseudo-op signifies that there is no more program code in the source file.

EQU - this evaluates the operand field and assigns the value to the label given at the beginning of the line. Absence of a label will be noted as an error. Note that equates with register names will allow the user to employ the new symbols as he would employ the corresponding register name.

EXT - defines a symbol to be external. This will enable later reference to these symbols by the linking loader at load time. The EXT pseudo op must be present in both the source in which the symbol is defined and the source in which the symbol is referenced. The label associated with EXT, then, is not processed as a normal label; it appears twice in the definition source.

HEADER - place a header at the top of the following pages. This header is located under the title generated by the TITLE pseudo-op. The string in the operand field (enclosed in single quotes) appears as the header.

LIST - turns on the program listing (generation of the listing file). This is the default.

MESSAGE - display a message on the user's terminal. This command is used to send the string in the operand field to the user's terminal. It may be used in conjunction with the DISPLAY pseudo-op to print a message along with the value displayed. The string in the operand field (enclosed in single quotes) appears as the message.

NOLIST - turns off program listing. The advantage of this pseudo-op is a slight increase in the efficiency of the assembler and lower printing costs.

NOSYM - inhibits the generation of a symbol table in the program listing. The generation of a symbol table is the default.

ONLY8080 - sets a switch which will raise an error if an instruction which will not execute on the 8080 is encountered. This option is disengaged as default.

ORG - sets the memory location counter of the assembler to the value of the expression in the operand field. If a label is present, it is also given that value. Secondly, this pseudo-op sets the address mode to absolute. The absolute addressing mode is the default. If an ORG is not specified, assembly will start with zero as the value of the memory location counter.

PAGE - forces a page eject. This will help in making listings more readable.

REL - sets the address mode to relative. No operand is required.

TITLE - places a title across the top of each page. The string in the operand field (enclosed in single quotes) appears as the title.

XREF - produces a cross-referenced symbol table at the end of the listing. The default is a normal, non-cross-referenced symbol table.

The following is an SDL description of these pseudo-ops.

```
<ALPHA>          : 'A' ! .. ! 'Z'
<ALPHANUM>       : <ALPHA> ! '0' ! .. ! '9'
<COLON>          : ':'
<LABEL>          : <ALPHA> <ALPHANUM>* <COLON>?
<HEX>            : 'A' ! .. ! 'F' ! '0' ! .. ! '9'
<HEXNUM>         : <HEX>+
<STRING>         : ''' "string of chapacters" '''
<B>              : ' '+
<B1>             : ' '*
<EXPRESSION>     : "a valid arithmetic or string expression for this
                   assembler (see text)"
<DIGIT>          : '0' ! .. ! '9'
<DECNUM>         : <DIGIT>+
<DB PSEUDO-OP>   : <LABEL>? <B> 'DB' <B> <EXPRESSION> (<B1> ',' <B1> <EXPRESSION>)*
<DEFB PSEUDO-OP> : <LABEL>? <B> 'DEFB' <B> <EXPRESSION>
<DEFM PSEUDO-OP> : <LABEL>? <B> 'DEFM' <B> ''' "a string" '''
<DEFS PSEUDO-OP> : <LABEL>? <B> 'DEFS' <B> <EXPRESSION>
<DEFW PSEUDO-OP> : <LABEL>? <B> 'DEFW' <B> <EXPRESSION>
<DISPLAY PSEUDO-OP> : <B> 'DISPLAY' <B> <EXPRESSION>
<DS PSEUDO-OP>   : <LABEL>? <B> 'DS' <B> <EXPRESSION>
<DW PSEUDO-OP>   : <LABEL>? <B> 'DW' <B> <EXPRESSION> (<B1> ',' <B1> <EXPRESSION>)*
<END PSEUDO-OP>  : <LABEL>? <B> 'END'
<EQU PSEUDO-OP>  : <LABEL> <B> 'EQU' <B> <EXPRESSION>
<EXT PSEUDO-OP>  : <LABEL> <B> 'EXT'
<HEADER PSEUDO-OP> : <LABEL>? <B> 'HEADER' <B> ''' <STRING> '''
<LIST PSEUDO-OP> : <B> 'LIST'
<MESSAGE PSEUDO-OP> : <B> 'MESSAGE' <B> ''' <STRING> '''
<NOLIST PSEUDO-OP> : <B> 'NOLIST'
<NOSYM PSEUDO-OP> : <B> 'NOSYM'
<ONLY8080 PSEUDO-OP> : <B> 'ONLY8080'
<ORG PSEUDO-OP>  : <LABEL>? <B> 'ORG' <B> <EXPRESSION>
<PAGE PSEUDO-OP> : <B> 'PAGE'
<REL PSEUDO-OP>  : <B> 'REL'
<TITLE PSEUDO-OP> : <B> 'TITLE' <B> ''' <STRING> '''
<XREF PSEUDO-OP> : <B> 'XREF'
```

2.2 Addressing Modes

There are two addressing modes supported by this assembler -- absolute and relative. If an entire assembly is run in absolute mode, then the resulting object module will be completely compatable with the ARIES/MUMS object format. To put the assembler in absolute mode, one simply uses the ORG pseudo-op, supplying the starting address of the memory location counter as the operand.

The relative addressing mode is more flexible. Using this mode, object libraries can be built using relocatable routines. The only problem encountered in relocation is the use of word-valued (two byte) addresses within the assembled program. In absolute mode, these addresses are indistinguishable from any pair of bytes, so relocation is impossible. In relative mode, these addresses are prefixed by the letter 'R' to signify a relative address. A two-byte value follows the 'R'; this is the offset from the beginning of the current relocatable module. Thus, when loading, this value is added to the value of the memory address at which the current code segment is to reside. Relative mode is entered by using the REL pseudo-op, which needs no operand. The default address mode of ASMZ80 is absolute.

Note that one can switch back and forth between absolute and relative addressing modes, referencing absolute symbols in relative code and relative symbols in absolute code. It is the responsibility of the programmer to be sure he is not overlaying some of his own code at load time if he chooses to change modes like this.

External symbols are provided to allow the programmer to use previously-assembled routines by name, rather than by the awkward method of using equates in which the programmer does the memory mapping. A symbol is declared external by the EXT pseudo-op, with the symbol at the beginning of the line being the label referenced. These externals can be either absolute or relative and must be resolved by the linker at load time.

2.3 Expression Syntax

Expressions analyzed by the assembler must be rather simple in form. Since parentheses are used for indirection, no effort was made to implement them in expressions. Hence, no nesting of expressions is permitted, and evaluation is done strictly from left to right with operator precedence. Also, no blanks are accepted within an expression, and addressing modes cannot be mixed freely. External reference symbols can't be used in expressions, and relative symbols can only be operated on by plus (+) and minus (-). The following operators are recognized by the assembler:

Operator	Priority	Description
*	4	Multiplication
/	4	Division
%	4	Modulo
+	3	Addition
-	3	Subtraction
&	2	Logical AND
!	1	Logical OR

2.4 Invoking the Assembler

Using the assembler is very simple once one has written the text file. Use

of the assembler is done in two steps: (1) initialization, which is done only once during a timesharing session, and (2) invocation of the assembler itself.

It is usually a good idea to initialize the user's timesharing environment immediately after logging in to the system. This is done by issuing the following INTERCOM commands:

```
ATTACH,P1,PROFIL,ID=XXXXXX
REWIND,P1
COPY,P1,PROFIL
RETURN,P1
RFL,64000
ETL,200
CONNECT,INPUT,OUTPUT
```

Once this has been done, the user is ready to perform his normal timesharing activities. When the user's program source file is ready to be assembled, the user executes ASMZ80 by issuing the following INTERCOM command:

```
BEGIN,ASSM,,SOURCE,LISTING,OBJECT,OBJECT2
```

SOURCE is the user's assembler language source file, LISTING is the assembler's output listing file, OBJECT is the assembler's INTEL-format output object file, and OBJECT2 is the assembler's ARIES-format output object file. All names are optional, the files Z80SRC, Z80LST, Z800BJ1, and Z800BJ2 being used if the replacement is not indicated. All files are rewound before and after using them.

The assembler will run in a field length of less than 20000B if the program is not large (less than 500 bytes), and has never (yet) run out of space with a field length of 64000B.

At the end of the assembly, the assembler will tell the user how many errors were detected. Errors are indicated in the output listing by a hash character ('#') in one of the first four columns of the error line. Hence, the user may use an editor to find the error lines by searching for this character in the first four columns. Also in the listing are the error message codes, located between the object code and actual source line (the actual source line is preceeded by a colon, ':').

An error-listing procedure is available to the user through the PROFIL. This is invoked by

```
BEGIN,ERRORS,,LISTING
```

where LISTING is the listing file generated by the assembly. This procedure will list all error lines on the user's terminal for his review.

CHAPTER 3

Use of the Linker

One of the functions of ASMZ80 is to allow the user to maintain object code libraries, removing the need to assemble all the routines he needs every time. To this goal, he must declare the names of his routines to be external when they are assembled and put the relocatable object into his library.

3.1 Library Files

The library file format is very simple, and it will allow the use of basic utilities to construct it. It is a segmented file of text, with each segment containing exactly one object module. For example, to add a newly-assembled object file, OBJ, to the library file, LIB, the user need only type:

```
REWIND,LIB
COPY,LIB,NEWLIB
REWIND,OBJ
COPY,OBJ,NEWLIB
REWIND,NEWLIB
REWIND,LIB
COPY,NEWLIB,LIB
```

This will add the object file to the library file, LIB.

3.2 Invoking the Linker

The procedure 'LINK' is used to run the linker. To invoke the linker, type:

```
BEGIN,LINK,,SOURCE,LIBRARY,OBJECT,OBJECT1,MESSAGE
```

where SOURCE is the main object module, LIBRARY is the object library file, OBJECT is the output object file in INTEL hex format, OBJECT1 is the output

object file in ARIES-format, and MESSAGE is the output message file produced by the linker which holds any diagnostics and a load map. SOURCE and LIBRARY must be relocatable object files in ARIES format.

When the linker starts running, the user will type a linker directive. The linker directives are:

R -- link and relocate. This tells the linker to leave the resolved object module in relocatable format.

A <address> - link and generate absolute code. This tells the linker to output the object in absolute format, with the hex string specifying the start address. This command contains no blanks, and takes the form of 'AHHHH', where 'H' represents any valid hexadecimal digit.

Any errors other than multiple symbol definition are considered to be catastrophic and will abnormally terminate the linker.

CHAPTER 4

Appendices

Appendix A
Summary of Z80 Mnemonics

```
ADC     A,Q8
ADC     HL,R16A

ADD     A,Q8
ADD     HL,R16A
ADD     IX,R16B
ADD     IY,R16C

AND     Q8

BIT     N3,Q8A

CALL    N16
CALL    CF,N16

CCF

CP      Q8
CPD
CPDR
CPI
CPIR

CPL

DAA

DEC     Q8A
DEC     R16D
```

Summary of Z80 Mnemonics, Con't

```
DI

DJNZ    E

EI

EX      SP,HL
EX      SP,IX
EX      SP,IY
EX      AF,AF'
EXX

HALT

IM      0
IM      1
IM      2

IN      A,(N8)
IN      R8,(C)

INC     Q8A
INC     R16D

IND
INDR
INI
INIR

JP      (HL)
JP      (IX)
JP      (IY)
JP      N16
JP      CF,N16

JR      E
JR      CFA,E

LD      R8,Q8
LD      A,(N16)
LD      (N16),A
LD      R16D,(N16)
LD      (N16),R16D
LD      R16D,N16
LD      SP,HL
LD      SP,IX
LD      SP,IY
LD      A,R
LD      A,I
LD      R,A
LD      I,A
LD      A,(BC)
LD      A,(DE)
LD      (BC),A
LD      (DE),A
LDD
LDDR
LDI
LDIR

NEG

NOP
```

Summary of Z80 Mnemonics, Con't

```
OR      Q8

OTDR
OTIR
OUTD
OUTI
OUT     (N8),A
OUT     (C),R8

POP     R16E

PUSH    R16E

RES     N3,Q8A

RET
RET     CF
RETI
RETN

RL      Q8A
RLA
RLC     Q8A

RLCA
RLD
RR      Q8A
RRA
RRC     Q8A
RRCA
RRD
RST     N3*8

SBC     A,Q8A
SBC     HL,R16A

SCF

SET     N3,Q8A

SLA     Q8A
SRA     Q8A
SRL     Q8A

SUB     Q8

XOR     Q8
```

Definition of Symbols

The special symbols in the above table have the following significance:

R8 -- any of the set (A,B,C,D,E,H,L)
Q8A -- any of R8 or [(HL),(IX+D),(IY+D)]
Q8 -- any of Q8A or N8
D -- when used in (IX+D) or (IY+D), it must have a value between -128 and +127 decimal, inclusive
E -- used with jump relative instructions, it must have a value between -126 and +129 decimal, inclusive. It is usually an expression of the form 'SYMBOL-$'.
R16 -- any of the set (AF,BC,DE,HL,IX,IY,SP)
R16A -- any of the set (BC,DE,HL,SP)
R16B -- any of the set (BC,DE,IX,SP)
R16C -- any of the set (BC,DE,IY,SP)
R16D -- any of the set (BC,DE,HL,IX,IY,SP)
R16E -- any of the set (AF,BC,DE,HL,IX,IY)
N3 -- a 3-bit integer; any of the set (0 .. 7)
N8 -- an 8-bit integer; any of the set (0 .. 255)
N16 -- a 16-bit integer; any of the set (0 .. 65535)
CF -- any of the set (Z,NZ,C,NC,PO,PE,M,P)
CFA -- any of the set (Z,NZ,C,NC)

Appendix B

Relocatable Object Format

This object file format is compatible with the MUMS object format implemented by Kominczak in the Universal Cross Assembler and the ARIES object format implemented by Conn in Project ARIES. This format additionally allows relocatable references by specifying an offset from the beginning of the current assembly and external references by specifying a name. All external symbols are listed in an External Symbol Dictionary (ESD) at the beginning of the object module to reduce lookahead problems. An object module can be described in SDL by the following:

```
<OBJECT MODULE>          : <ESD BLOCK>* <DATA BLOCK>* '$'

<ESD BLOCK>              : '#S' <SYMBOL ENTRY>+ '&' <V8>

<SYMBOL ENTRY>           : <RELATIVE DEFINITION> !
                           <ABSOLUTE DEFINITION> !
                           <REFERENCE DEFINITION>

<RELATIVE DEFINITION>    : 'R' <NAME> <V16>

<ABSOLUTE DEFINITION>    : 'A' <NAME> <V16>

<REFERENCE DEFINITION>   : 'F' <NAME>

<NAME>                   : <ALPHA> <ALPHANUM>* <BLANK>

<ALPHA>                  : 'A' ! .. ! 'Z'

<ALPHANUM>               : <ALPHA> ! '0' ! .. ! '9'

<BLANK>                  : "space character"

<V8>                     : <HEX DIGIT> <HEX DIGIT>

<V16>                    : <V8> <V8>

<HEX DIGIT>              : 'A' ! .. ! 'F' ! '0' ! .. ! '9'

<DATA BLOCK>             : '#' <V16> <VALUE>+ '&' <V8>

<UALTE>                  : <U8> !
                           <RELATIVE REFERENCE> !
                           <EXTERNAL REFERENCE>

<RELATIVE REFERENCE>     : 'R' <V16>
```

<EXTERNAL REFERENCE> : 'X' <NAME>

All characters except '#', '&', and the checksum contribute to the modulo-256 checksum in the following manner:

<V8> -- the ASCII characters are converted to a single byte value, and the value of the byte is added to the checksum.

<ALPHA> -- the value is computed by adding the ordinal number ('A'=1) to 9. For example, for the character 'Z', 35 will be added to the checksum.

All other characters, including blank, are given the value zero.

All <V16>'s are not equal under the eyes of the linker. When in absolute load mode, the address immediately following the beginning hash character ('#') is given high-order byte first. All other word values are given low-order byte first.

Appendix C

Assembler Error Messages

A -- improper addressing mode usage in an expression
E -- null addressing mode encountered in an expression; this should never occur
F -- forward reference detected in operand of an EQU or DS
I -- invalid instruction mnemonic
M -- multiple definitions of a symbol have been encountered
N -- invalid character encountered in conversion of a numeric operand
O -- the all-inclusive operand error. This usually signifies improper usage of predefined register and flag names.
P -- missing 'END' statement
Q -- an equate cannot be made to an indirect symbol
R -- the range of a jump relative (-126 to +129 from the current instruction) has been exceeded
S -- a string has not been terminated
U -- an undefined symbol has been used in an operand field
Z -- an instruction has been used which is valid for the Z80 but not for the 8080. This error can occur only if the 'ONLY8080' pseudo-op has been used.

Appendix D

References

Conn, Richard, "ARIAN -- An Implementation of a Microcomputer Operating System", May 78; Master's Thesis, filed with the Graduate College, University of Illinois at Urbana-Champaign, Urbana, IL 61801. Also available through Defense Documentation Center, Alexandria, VA, under AD Number ADA060210.

Conn, Richard, "The Monitor Command System User's Manual", Feb 78; Dept of Computer Science Technical Report Number UIUCDCS-R-78-912, Dept of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

INTEL Corporation, "INTEL 8080 Microcomputer Systems User's Manual", Sep 75; INTEL Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

MOSTEK Corporation, "MK 3880 Central Processing Unit", May 78; MOSTEK Corporation, 1215 W. Crosby Road, Carrollton, TX 75006.

MOSTEK Corporation, "Z80 Programming Manual", Dec 77; MOSTEK Corporation, 1215 W. Crosby Road, Carrollton, TX 75006.

ZILOG Corporation, "Z80-Assembly Language Programming Manual", Jan 78; ZILOG Corporation, 10460 Bubb Road, Cupertino, CA 95014.

CHAPTER 5

Source Code Listings

This chapter presents the source code listings of the Z80ASM system. Specifically, the programs addressed by this chapter are --

ZLDR	The Relocatable Linking Loader for the Z80 Assembler
Z80ASM	The Z80 Assembler

Z80ASM - Z80 CROSS ASSEMBLER and LINKER USER'S MANUAL

Source Code Listings

ZLDR -- The Relocatable Linking Loader

```
(*ZLDR*)
(*  ZLDR WAS ORIGINALLY WRITTEN BY GEORGE LEHMANN AT THE *)
(*    UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN, IL     *)
(*  ZLDR IS MODIFIED AND MAINTAINED ON THE ARRADCOM CDC  *)
(*    6500/6600 BY 1LT RICHARD CONN, MISD, SATCOMA, FORT *)
(*    MONMOUTH, NJ                                       *)
(*********************************************************)
(*  THIS IS A LOADER. IT WILL NOT GET YOU LOADED. HOWEVER, IT *)
(*  WILL ALLOW YOU, THE USER, TO COMBINE SEVERAL SEPERATE     *)
(*  ASSEMBLIES INTO ONE GIANT LOAD MODULE.                    *)
(*********************************************************)

PROGRAM Z80LDR (INPUT,OUTPUT,LST,PRMFILE,OBJ,OBJLIB);

CONST
   MXSYM = 100;
   OBJMAX= 68;

TYPE
   ISNUM    = 0..MXSYM;
   WORD     = 0..65535;
   OBJTYPE = (BYTE,AWORD,RWORD,XWORD,ORG,REL,SYM);
   OBJEL=PACKED RECORD
         OTYPE : OBJTYPE;
         VAL   : WORD;
         END;
   LIBRARY = SEGMENTED FILE OF CHAR;
   ADRTYPE = (NULL,ABSOLUTE,RELATIVE);
   STEPTR  = ^STE;
   STE = RECORD
         NAME  : ALFA;
         VALUE : INTEGER;
         ADM   : ADRTYPE;
         RESLVD: BOOLEAN;
         ISN   : ISNUM;
         LSON  : STEPTR;
         RSON  : STEPTR;
         END;

VAR
   LST,OBJ,PRMFILE : TEXT;
   TMPOBJ          : FILE OF OBJEL;
   OBJLIB          : LIBRARY;
   MAP             : BOOLEAN;
   PC,RELBASE      : INTEGER;
   LDMODE,OUTMODE  : ADRTYPE;
   STHEAD,TMPHEAD  : STEPTR;
   INCKSUM,OUTCKSUM: INTEGER;
   STFOUND         : STEPTR;
   ISNS            : ARRAY [1..MXSYM] OF STEPTR;
   LASTISN         : ISNUM;
   OBJCNT,START    : INTEGER;
   HAS             : CHAR;

(*********************************************************)
```

```
(*  PROCEDURE ERROR SIMPLY PRINTS ERROR MESSAGES. SOME ERRORS *)
(*  MAY DO SOME SPECIAL PROCESSING OR EVEN HALT. THIS IS TAKEN*)
(*  CARE OF IN THE INDIVIDUAL CASE BRANCHES                   *)
(*************************************************************)

   PROCEDURE ERROR (IDX : INTEGER);
   BEGIN
      WRITE(LST,' ERROR - ');
      CASE IDX OF
      1 : WRITELN(LST,'IMPROPER ESD BLOCK IN PRIMARY FILE');
      2 : WRITELN(LST,'ESD CHECKSUM ERROR IN PRIMARY FILE');
      3 : WRITELN(LST,'ILLEGAL CHARACTER IN HEX CONVERT');
      4 : WRITELN(LST,'IMPROPER CHARACTER IN PRIMARY FILE DATA BLOCK');
      5 : WRITELN(LST,'DATA BLOCK CKSUM ERROR IN PRIMARY FILE');
      6 : WRITELN(LST,'IMPROPER TERMINATOR IN PRIMARY FILE');
      7 : WRITELN(LST,'MAXIMUM NUMBER OF EXTERNAL SYMBOLS (',MXSYM:5,
            'HAS BEEN EXCEEDED');
      8 : WRITELN(LST,'EXTERNAL SYMBOL IN DATA BLOCK NOT DEFINED IN ',
            'SYMBOL BLOCK');
      9 : WRITELN(LST,'IMPROPER ESD ENTRY IN LIBRARY FILE');
      10: WRITELN(LST,'MAXIMUM NUMBER OF TEMPORARY SYMBOLS EXCEEDED');
      END;
      HALT;
   END;

(**********(*********************************************)
(*  FUNCTION HEX RETURNS AN ASCII CHARACTER REPRESENTING THE  *)
(*  INPUT PARAMETER. ONLY THE FOUR LEAST SIGNIFICANT BITS ARE *)
(*  USED.                                                     *)
(*************************************************************)

   FUNCTION HEX (I : INTEGER) : CHAR;
   BEGIN
      I := I MOD 16;
      IF I<0 THEN I := I + 16;
      IF (I IN [0..9]) THEN HEX := CHR(I+ORD('0'))
      ELSE HEX := CHR(I-10+ORD('A'))
   END;

(*************************************************************)
(*  FUNCTION VAL RETURNS THE INTEGER VALUE OF A HEXADECIMAL   *)
(*  ASCII CHARACTER.                                          *)
(*************************************************************)

   FUNCTION VAL (CH : CHAR) : INTEGER;
   BEGIN
      IF (CH IN ['0'..'9']) THEN VAL := ORD(CH) - ORD('0')
      ELSE IF (CH IN ['A'..'F']) THEN VAL := ORD(CH) + 10 - ORD('A')
      ELSE BEGIN
         ERROR(3);
         VAL := 0
         END
   END;

(*************************************************************)
```

```
(*  PROCEDURE INIT DOES JUST THAT, IT INITIALIZES.                *)
(*****************************************************************)

   PROCEDURE INIT;
   VAR
      CH  :  CHAR;
   BEGIN
      RESET(INPUT);
      LDMODE:= RELATIVE;
      MAP   := TRUE;
      START := 0;
         READ(CH);
         IF (CH='R') THEN
           BEGIN
             LDMODE := RELATIVE;
             WRITELN(' RELATIVE LINK');
           END
         ELSE IF (CH='A') THEN
            BEGIN
            WRITE(' ABSOLUTE LINK AT ');
            LDMODE := ABSOLUTE;
            START := 0;
            READ(CH);
            WHILE (CH IN ''0'..'9','A'..'F'!) DO
               BEGIN
               WRITE(CH);
               START := START*16 + VAL(CH);
               READ(CH);
               END;
            END;
      WRITELN;
      PC := START;
      LASTISN := 0;
      NEW(STHEAD);
      STHEAD~.NAME := '          ';
      STHEAD~.LSON := NIL;
      STHEAD~.RSON := NIL;
      STHEAD~.RESLVD := TRUE;
      RESET(PRMFILE);
      REWRITE(LST);
      REWRITE(TMPOBJ);
   END;

(*****************************************************************)
(*  FUNCTION UNRESOLVED RETURNS TRUE IF AN EXTERNAL REFERENCE *)
(*  WHICH HAS NOT BEEN RESOLVED IS FOUND IN THE SYMBOL TABLE. *)
(*****************************************************************)

   FUNCTION UNRESOLVED : BOOLEAN;
      FUNCTION UNRES2 (ST : STEPTR) : BOOLEAN;
      VAR URFOUND : BOOLEAN;  (* UNRESOLVED REFERENCE FOUND *)
      BEGIN
         WITH ST~ DO BEGIN
         (* CHECK PARENT NODE *)
         URFOUND := NOT RESLVD;
```

```
        (* IF PARENT IS RESOLVED, CHECK LEFT SON *)
        IF (NOT URFOUND) AND (LSON<>NIL) THEN URFOUND := UNRES2(LSON);
        (* IF LEFT SON TREE IS RESOLVED, CHECK RIGHT SON *)
        IF (NOT URFOUND) AND (RSON<>NIL) THEN URFOUND := UNRES2(RSON);
        UNRES2 := URFOUND
        END
     END;
  BEGIN
     UNRESOLVED := UNRES2(STHEAD)
  END;

(*************************************************************)
(*  FUNCTION SRCHST SEARCHES THE SPECIFIED SYMBOL TABLE FOR  *)
(*  THE SPECIFIED SYMBOL. IF FOUND, TRUE IS RETURNED AND     *)
(*  STFOUND POINTS TO THE CORRECT SYMBOL TABLE ENTRY. IF NOT *)
(*  FOUND, FALSE IS RETURNED AND STFOUND POINTS TO THE ENTRY *)
(*  TO WHICH THE NEW ENTRY SHOULD BE APPENDED.               *)
(*************************************************************)

  FUNCTION SRCHST (SYM : ALFA; STEL : STEPTR) : BOOLEAN;
  BEGIN
     STFOUND := STEL; (* LEAVE POINTER TRAIL *)
     WITH STEL^ DO
        BEGIN
        IF (SYM=NAME) THEN SRCHST := TRUE
        ELSE IF (SYM<NAME) THEN
           IF (LSON<>NIL) THEN SRCHST := SRCHST(SYM,LSON)
           ELSE SRCHST := FALSE
        ELSE IF (RSON<>NIL) THEN SRCHST := SRCHST(SYM,RSON)
           ELSE SRCHST := FALSE
        END
  END;

(*************************************************************)
(*  PROCEDURE INSST INSERTS A SYMBOL INTO A SYMBOL TABLE.    *)
(*  THE TABLE IS VARIABLE TO SUPPORT TEMPORARY TABLES DURING *)
(*  THE LIBRARY SEARCH.                                      *)
(*************************************************************)

  PROCEDURE INSST (SYM : ALFA; ADDR : INTEGER;
                   DEFINED : BOOLEAN; ST : STEPTR; MODE : ADRTYPE);
  VAR PTR : STEPTR;
  BEGIN
     WITH ST^ DO
        IF SRCHST(SYM,ST) THEN
           BEGIN
           IF (DEFINED AND STFOUND^.RESLVD) THEN
              WRITELN(LST,' ERROR - MULTIPLE DEFINITIONS ENCOUNTERED FOR ',SYM)
           ELSE IF DEFINED THEN
              BEGIN
              STFOUND^.RESLVD := TRUE;
              STFOUND^.ADM    := MODE;
              STFOUND^.VALUE  := ADDR
              END
           END
```

```
         ELSE BEGIN
            NEW(PTR);
            PTR~.NAME  := SYM;
            PTR~.VALUE := ADDR;
            PTR~.RESLVD:= DEFINED;
            PTR~.ADM   := MODE;
            PTR~.LSON  := NIL;
            PTR~.RSON  := NIL;
            IF (LASTISN=MXSYM) THEN ERROR(7)
            ELSE BEGIN
               LASTISN := LASTISN + 1;
               ISNS`LASTISN! := PTR;
               PTR~.ISN := LASTISN;
               END;
            IF (SYM<STFOUND~.NAME) THEN STFOUND~.LSON := PTR
            ELSE STFOUND~.RSON := PTR
            END
   END;

(*************************************************************)
(*   PROCEDURE WRTMP WRITES AN OBJECT ELEMENT INTO THE TEMPORARY
(*   OBJECT FILE.                                            *)
(*************************************************************)

   PROCEDURE WRTMP (OT : OBJTYPE; V : WORD);
   VAR   X : OBJEL;
   BEGIN
      X.OTYPE := OT;
      X.VAL   := V;
      WRITE(TMPOBJ,X);
   END;

(*************************************************************)
(*  PROCEDURE OUTMAP PRINTS THE LOADER MAP IF REQUESTED.     *)
(*************************************************************)

   PROCEDURE OUTMAP;
   VAR MAPCOUNT : INTEGER;
      PROCEDURE OUTMAP2 (ST : STEPTR);
      BEGIN
         WITH ST~ DO BEGIN
         IF LSON<>NIL THEN OUTMAP2(LSON);
         IF NAME<>'          ' THEN
            BEGIN
            WRITE(LST,' ',NAME,' ');
            IF RESLVD THEN WRITE(LST,HEX(VALUE DIV 4096),HEX(VALUE DIV 256),
                           HEX(VALUE DIV 16),HEX(VALUE),'       ')
            ELSE WRITE(LST,'UNRESOLVED ');
            MAPCOUNT := MAPCOUNT + 1;
            IF MAPCOUNT = 3 THEN
               BEGIN
               WRITELN(LST);
               MAPCOUNT := 0
               END
            END;
```

```
      IF RSON<>NIL THEN OUTMAP2(RSON)
      END
   END;

BEGIN
   MAPCOUNT := 0;
   WRITELN(LST);
   OUTMAP2(STHEAD);
   WRITELN(LST);
END;

{*************************************************************}
{*  PROCEDURE LDPRIMFILE WILL LOAD THE PRIMARY FILE. IF ANY   *}
{*  EXTERNAL REFERENCES ARE NO IN THE SYMBOL TABLE, A FLAG IS *}
{*  IS SET TO INDICATE A LIBRARY SEARCH IS NEEDED.            *}
{*************************************************************}

PROCEDURE LDPRIMFILE;
VAR
   NAME    : ALFA;
   VALUE, RELPC, I     : INTEGER;
   CH,CH2,C1,C2,C3,C4 : CHAR;
   LINEDONE, RELLOAD   : BOOLEAN;

   {*************************************************************}
   {*  PROCEDURE READPRM READS ONE CHARACTER FROM THE PRIMARY    *}
   {*  FILE, AND UPDATES THE CHECKSUM ACCORDINGLY.               *}
   {*************************************************************}

   PROCEDURE READPRM ( VAR CH : CHAR);
   BEGIN
      READ(PRMFILE,CH);
      IF (CH IN ['0'..'9']) THEN INCKSUM := INCKSUM + ORD(CH) - ORD('0')
      ELSE IF (CH IN ['A'..'Z']) THEN INCKSUM := INCKSUM + ORD(CH) - ORD('A') + 10
   END;

   {*************************************************************}
   {*  PROCEDURE GETNAME READS AN ESD NAME ENTRY AND PACKS IT    *}
   {*  INTO VARIABLE 'NAME'.                                     *}
   {*************************************************************}

   PROCEDURE GETNAME;
   VAR SYM : ARRAY[1..11] OF CHAR;
       I   : INTEGER;
   BEGIN
      FOR I := 1 TO 10 DO SYM[I] := ' ';
      I := 1;
      REPEAT
         READPRM(SYM[I]);
         I := I + 1
         UNTIL SYM[I-1]=' ';
      PACK(SYM,1,NAME);
   END;

   {*************************************************************}
```

```
(*  FUNCTION GETVAL READS A WORD VALUE FROM THE PRIMARY FILE)
(*  AND UPDATES THE CHECKSUM.                                *)
(************************************************************)

   FUNCTION GETVAL : INTEGER;
   VAR
      I            : INTEGER;
   BEGIN
      READ(PRMFILE,C1,C2,C3,C4);
      I := VAL(C1)*16 + VAL(C2) + VAL(C3)*4096 + VAL(C4)*256;
      INCKSUM := INCKSUM + I + I DIV 256;
      GETVAL := I
   END;

BEGIN
   (* LOAD PRIMARY SYMBOL TABLE, IF ANY *)
   INCKSUM := 0;
   RELBASE := PC;
   REPEAT READ(PRMFILE,CH)
      UNTIL (CH=HAS) OR (CH='$');
   WHILE (PRMFILE~='S') DO
      BEGIN
      READPRM(CH); (* DISCARD 'S' *)
      LINEDONE := FALSE;
      REPEAT
         READPRM(CH); (* GET INDICATOR *)
         IF NOT (CH IN ['A','R','F','&']) THEN ERROR(1)
         ELSE CASE CH OF
         'A' : BEGIN
               GETNAME;
               VALUE := GETVAL;
               INSST(NAME,VALUE,TRUE,STHEAD,ABSOLUTE)
               END;
         'R' : BEGIN
               GETNAME;
               VALUE := GETVAL + RELBASE;
               INSST(NAME,VALUE,TRUE,STHEAD,LDMODE)
               END;
         'F' : BEGIN
               GETNAME;
               INSST(NAME,0,FALSE,STHEAD,NULL)
               END;
         '&' : BEGIN
               READ(PRMFILE,CH,CH2);
               VALUE := VAL(CH)*16 + VAL(CH2);
               IF VALUE <> (INCKSUM  MOD 256) THEN ERROR(2);
               LINEDONE := TRUE;
               INCKSUM  := 0;
               READLN(PRMFILE);
               REPEAT READ(PRMFILE,CH)
                  UNTIL (CH=HAS) OR (CH='$');
               END;
         END (* END CASE STATEMENT *)
         UNTIL LINEDONE;
      END;  (* END SYMBOL TABLE PROCESSING *)
```

```
WHILE (CH=HAS) DO
   BEGIN
   RELLOAD := PRMFILE^='R';
   IF RELLOAD THEN
      BEGIN
      READPRM(CH);
      WRTMP(REL,0)
      END
   ELSE BEGIN
      READ(PRMFILE,C1,C2,C3,C4);
      RELPC := PC;
      PC := VAL(C1)*4096 + VAL(C2)*256 + VAL(C3)*16 + VAL(C4);
      INCKSUM := PC + PC DIV 256;
      WRTMP(ORG,PC);
      END;
   LINEDONE := FALSE;
   REPEAT
      READ(PRMFILE,CH);
      IF NOT(CH IN ['0'..'9','A'..'F','R','X','&']) THEN ERROR(4)
      ELSE CASE CH OF
      '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F':
         BEGIN
         READ(PRMFILE,CH2);
         I := VAL(CH)*16 + VAL(CH2);
         INCKSUM := INCKSUM + I;
         WRTMP (BYTE,I);
         PC := PC + 1;
         END;
      'R' : BEGIN
         INCKSUM := INCKSUM+ORD('0');
         I := GETVAL + RELBASE;
         WRTMP (RWORD,I);
         PC := PC + 2;
         END;
      'X' : BEGIN
         INCKSUM := INCKSUM + 33;
         GETNAME;
         IF SRCHST(NAME,STHEAD) THEN
            WITH STFOUND^ DO
            IF RESLVD THEN
               IF (ADM=RELATIVE) THEN WRTMP(RWORD,VALUE)
               ELSE WRTMP(AWORD,VALUE)
            ELSE WRTMP (XWORD,ISN)
         ELSE ERROR(8);
         PC := PC + 2;
         END;
      '&' : BEGIN
         READ(PRMFILE,CH,CH2);
         I := VAL(CH)*16 + VAL(CH2);
         IF I<>INCKSUM MOD 256 THEN ERROR(5);
         READLN(PRMFILE);
         LINEDONE := TRUE;
         IF NOT RELLOAD THEN PC := RELPC;
         REPEAT READ(PRMFILE,CH) UNTIL (CH=HAS) OR (CH='$');
         INCKSUM := 0;
```

```
            OUTCKSUM:= 0;
            END;
         END; (* END CASE STATEMENT *)
         UNTIL LINEDONE; (* ONE LOAD BLOCK DONE *)
      END; (* END DATA BLOCK PROCESSING *)
   IF CH<>'$' THEN ERROR(6);
END;

(*************************************************************)
(*  PROCEDURE RESOLVE SEARCHES THE LIBRARY FOR DEFINITIONS OF *)
(*  THE REFERENCES FOUND IN THE PRIMARY FILE. IF A LIBRARY    *)
(*  ROUTINE PRODUCES UNRESOLVED REFERENCES, ANOTHER SEARCH    *)
(*  THROUGH THE LIBRARY WILL BE MADE. THE SEARCH IS ENDED WHEN*)
(*  EITHER ALL REFERENCES ARE RESOLVED, OR ONE PASS IS MADE   *)
(*  THROUGH THE LIBRARY WITHOUT FINDING ANY NEEDED DEFINITIONS*)
(*************************************************************)

PROCEDURE RESOLVE;
CONST
   MXTMPS = 100;

VAR
   TMPSYM : ARRAY [1..MXTMPS] OF STE;
   SYMCNT : 0..MXTMPS;
   NOFIND : BOOLEAN;
   ONEPASS: BOOLEAN;
   I      : INTEGER;

   NAME : ALFA;
   CH,CH2,C1,C2,C3,C4 : CHAR;
   LINEDONE : BOOLEAN;
   VALUE    : INTEGER;

(*************************************************************)
(*   PROCEDURE INSTMP INSERTS A SYMBOL INTO THE TEMPORARY    *)
(*   SYMBOL TABLE.                                           *)
(*************************************************************)

PROCEDURE INSTMP (S : ALFA; V : INTEGER; DEF : BOOLEAN; M : ADRTYPE);
BEGIN
   IF SYMCNT=MXTMPS THEN ERROR(10);
   SYMCNT := SYMCNT + 1;
   WITH TMPSYM[SYMCNT] DO
      BEGIN
      NAME := S;
      VALUE:= V;
      RESLVD := DEF;
      ADM  := M;
      END;
END;

(*************************************************************)
(*  PROCEDURE READOBJ READS ONE CHARACTER FROM THE LIBRARY    *)
(*  FILE, AND UPDATES THE CHECKSUM ACCORDINGLY.               *)
(*************************************************************)
```

```
   PROCEDURE READOBJ ( VAR CH : CHAR);
   BEGIN
      READ(OBJLIB,CH);
      IF (CH IN ''0'..'9'!) THEN INCKSUM := INCKSUM + ORD(CH) - ORD('0')
      ELSE IF (CH IN ''A'..'Z'!) THEN INCKSUM := INCKSUM + ORD(CH) - ORD('A') + 10
   END;

(*************************************************************)
(*  PROCEDURE GETNAME READS AN ESD NAME ENTRY AND PACKS IT   *)
(*  INTO VARIABLE 'NAME'.                                    *)
(*************************************************************)

   PROCEDURE GETNAME;
   VAR SYM : ARRAY`1..11! OF CHAR;
       I   : INTEGER;
   BEGIN
      FOR I := 1 TO 10 DO SYM`I! := ' ';
      I := 1;
      REPEAT
         READOBJ(SYM`I!);
         I := I + 1
         UNTIL SYM`I-1!=' ';
      PACK(SYM,1,NAME);
   END;

(*************************************************************)
(*  FUNCTION GETVAL READS A WORD VALUE FROM THE LIBRARY FILE  *)
(*  AND UPDATES THE CHECKSUM.                                *)
(*************************************************************)

   FUNCTION GETVAL : INTEGER;
   VAR
       I            : INTEGER;
   BEGIN
      READ(OBJLIB,C1,C2,C3,C4);
      I := VAL(C1)*16 + VAL(C2) + VAL(C3)*4096 + VAL(C4)*256;
      INCKSUM := INCKSUM + I + I DIV 256;
      GETVAL := I
   END;
(*************************************************************)
(*   PROCEDURE LOADSYMS READS THE ESD FROM THE NEXT LIBRARY  *)
(*   PARTITION.                                              *)
(*************************************************************)

   PROCEDURE LOADSYMS;
   BEGIN
      SYMCNT := 0;
      INCKSUM := 0;
      RELBASE := PC;
      REPEAT READ(OBJLIB,CH)
         UNTIL (CH=HAS) OR (CH='$');
      WHILE (OBJLIB<>'S') DO
         BEGIN
         READOBJ(CH); (* DISCARD 'S' *)
```

```
        LINEDONE := FALSE;
        REPEAT
           READOBJ(CH); (* GET INDICATOR *)
           IF NOT (CH IN ['A','R','F','&']) THEN ERROR(9)
           ELSE CASE CH OF
           'A' : BEGIN
                 GETNAME;
                 VALUE := GETVAL;
                 INSTMP(NAME,VALUE,TRUE,ABSOLUTE)
                 END;
           'R' : BEGIN
                 GETNAME;
                 VALUE := GETVAL + RELBASE;
                 INSTMP(NAME,VALUE,TRUE,LDMODE)
                 END;
           'F' : BEGIN
                 GETNAME;
                 INSTMP(NAME,0,FALSE,NULL)
                 END;
           '&' : BEGIN
                 READ(OBJLIB,CH,CH2);
                 VALUE := VAL(CH)*16 + VAL(CH2);
                 IF VALUE <> (INCKSUM  MOD 256) THEN ERROR(2);
                 LINEDONE := TRUE;
                 INCKSUM  := 0;
                 READLN(OBJLIB);
                 REPEAT READ(OBJLIB,CH)
                    UNTIL (CH=HAS) OR (CH='$');
                 END;
           END (* END CASE STATEMENT *)
           UNTIL LINEDONE;
        END; (* END SYMBOL TABLE PROCESSING *)
   END;

(*************************************************************)
(*   FUNCTION URINLIST CHECKS TO SEE IF ONE OF THE TEMPORARY  *)
(*   SYMBOLS WILL FILL AN UNRESOLVED ENTRY IN THE MAIN TABLE  *)
(*   IF SO, TRUE IS RETURNED.                                 *)
(*************************************************************)

FUNCTION URINLIST : BOOLEAN;
VAR
   I       : INTEGER;
   NOMATCH : BOOLEAN;
BEGIN
   NOMATCH := TRUE;
   I := 0;
   WHILE (NOMATCH) AND (I<SYMCNT) DO
      BEGIN
      I := I + 1;
      IF SRCHST(TMPSYM^[I].NAME,STHEAD) THEN
         IF NOT (STFOUND^.RESLVD) THEN NOMATCH := FALSE;
      END;
   URINLIST := NOT NOMATCH;
END;
```

```
{*************************************************************}
{*   PROCEDURE LOADDATA COPIES THE OBJECT PARTITION INTO THE  *}
{*   TEMPORARY FILE.                                         *}
{*************************************************************}
   PROCEDURE LOADDATA;
   VAR
      RELPC,I              : INTEGER;
      LINEDONE,RELLOAD     : BOOLEAN;
   BEGIN
   WHILE (CH=HAS) DO
      BEGIN
      RELLOAD := OBJLIB^='R';
      IF RELLOAD THEN
         BEGIN
         READOBJ(CH);
         WRTMP(REL,0)
         END
      ELSE BEGIN
         READ(OBJLIB,C1,C2,C3,C4);
         RELPC := PC;
         PC := VAL(C1)*4096 + VAL(C2)*256 + VAL(C3)*16 + VAL(C4);
         INCKSUM := PC + PC DIV 256;
         WRTMP(ORG,PC);
         END;
      LINEDONE := FALSE;
      REPEAT
         READ(OBJLIB,CH);
         IF NOT(CH IN ['0'..'9','A'..'F','R','X','&']) THEN ERROR(4)
         ELSE CASE CH OF
         '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F':
            BEGIN
            READ(OBJLIB,CH2);
            I := VAL(CH)*16 + VAL(CH2);
            INCKSUM := INCKSUM + I;
            WRTMP (BYTE,I);
            PC := PC + 1;
            END;
         'R' : BEGIN
            INCKSUM := INCKSUM+ORD('0');
            I := GETVAL + RELBASE;
            WRTMP (RWORD,I);
            PC := PC + 2;
            END;
         'X' : BEGIN
            INCKSUM := INCKSUM + 33;
            GETNAME;
            IF SRCHST(NAME,STHEAD) THEN
               WITH STFOUND^ DO
               IF RESLVD THEN
                  IF (ADM=RELATIVE) THEN WRTMP(RWORD,VALUE)
                  ELSE WRTMP(AWORD,VALUE)
               ELSE WRTMP (XWORD,ISN)
            ELSE ERROR(8);
            PC := PC + 2;
```

```
                END;
             '&' : BEGIN
                READ(OBJLIB,CH,CH2);
                I := VAL(CH)*16 + VAL(CH2);
                IF I<>INCKSUM MOD 256 THEN ERROR(5);
                READLN(OBJLIB);
                LINEDONE := TRUE;
                IF NOT RELLOAD THEN PC := RELPC;
                REPEAT READ(OBJLIB,CH) UNTIL (CH=HAS) OR (CH='$');
                INCKSUM := 0;
                OUTCKSUM:= 0;
                END;
             END; (* END CASE STATEMENT *)
             UNTIL LINEDONE; (* ONE LOAD BLOCK DONE *)
          END; (* END DATA BLOCK PROCESSING *)
       IF CH<>'$' THEN ERROR(6)
       ELSE GETSEG(OBJLIB);
       END;

    BEGIN
       RESET(OBJLIB);
       ONEPASS := FALSE;
       NOFIND  := TRUE;
       WHILE (NOT ONEPASS) AND UNRESOLVED DO
          IF EOF(OBJLIB) THEN
             BEGIN
             ONEPASS := NOFIND;
             NOFIND := TRUE;
             RESET(TMPOBJ);
             END
          ELSE BEGIN
             LOADSYMS;
             IF URINLIST THEN
                BEGIN
                FOR I := 1 TO SYMCNT DO
                   WITH TMPSYM^[I] DO INSST(NAME,VALUE,RESLVD,STHEAD,ADM);
                NOFIND := FALSE;
                LOADDATA;
                END
             ELSE GETSEG(OBJLIB);
             END
    END;

(*****************************************************************)
(*   PROCEDURE FLUSH OUTPUTS THE CHECKSUM INTO THE OBJECT FILE   *)
(*****************************************************************)

    PROCEDURE FLUSH;
    BEGIN
       WRITELN(OBJ,'&',HEX(OUTCKSUM DIV 16),HEX(OUTCKSUM));
       OBJCNT    := 0;
       OUTCKSUM := 0;
    END;

(*****************************************************************)
```

```
(*  PROCEDURE EMIT PUTS A SINGLE BYTE INTO THE OUTPUT OBJECT  *)
(*  FILE.                                                     *)
(************************************************************)

   PROCEDURE EMIT( V : INTEGER);
   BEGIN
      IF OBJCNT>OBJMAX THEN FLUSH;
      IF OBJCNT = 0 THEN
         BEGIN
         IF LDMODE=RELATIVE THEN
            BEGIN
            WRITE(OBJ,HAS,'R');
            OUTCKSUM := ORD('0');
            OBJCNT := 2;
            END
         ELSE
            BEGIN
            WRITE(OBJ,HAS,HEX(PC DIV 4096),HEX(PC DIV 256),HEX(PC DIV 16),
               HEX(PC));
            OUTCKSUM := PC + PC DIV 256;
            OBJCNT := 5;
            END;
         END;
      WRITE(OBJ,HEX(V DIV 16),HEX(V));
      OUTCKSUM := OUTCKSUM + V;
      PC := PC + 1;
      OBJCNT   := OBJCNT + 2;
   END;
(************************************************************)
(*   PROCEDURE EMITW PUTS A WORD INTO THE OUTPUT OBJECT FILE  *)
(*   IN HEX CHARACTER FORMAT.                                 *)
(************************************************************)

   PROCEDURE EMITW(V : INTEGER; M : ADRTYPE);
   BEGIN
      IF (M=RELATIVE) THEN
         BEGIN
         IF OBJCNT>OBJMAX-5 THEN FLUSH;
         IF OBJCNT=0 THEN
            BEGIN
            WRITE(OBJ,HAS,'R');
            OUTCKSUM := ORD('0');
            OBJCNT := 2;
            END;
         WRITE(OBJ,'R');
         OUTCKSUM := OUTCKSUM + ORD('0');
         OBJCNT := OBJCNT + 1;
         END;
      EMIT(V);
      EMIT(V DIV 256);
   END;

(************************************************************)
(*  PROCEDURE PASSTWO WILL COPY THE TEMPORARY OBJECT FILE TO  *)
(*  THE OUTPUT OBJECT FILE, COMPLETING RESOLUTION OF ALL      *)
```

```
(*  REFERENCES. THIS ROUTINE WILL BE CALLED ONLY IF DEFINITIONS)
(*  HAVE BEEN FOUND FOR ALL EXTERNAL REFERENCES.              *)
(*************************************************************)

   PROCEDURE PASSTWO;
   VAR CH : CHAR;
       X  : OBJEL
     ·RELPC : INTEGER;
   BEGIN
      RESET(TMPOBJ);
      REWRITE(OBJ);
      OUTMODE := RELATIVE;
      PC      := START;
      RELPC   := START;
      WHILE NOT EOF(TMPOBJ) DO
         BEGIN
         READ(TMPOBJ,X);
         CASE X.OTYPE OF
            BYTE : EMIT(X.VAL);
            AWORD: EMITW(X.VAL,ABSOLUTE);
            RWORD: EMITW(X.VAL,LDMODE);
            XWORD: WITH ISNS'X.VAL!~ DO
                      IF ADM=RELATIVE THEN EMITW(VALUE,LDMODE)
                      ELSE EMITW(VALUE,ABSOLUTE);
            REL  : BEGIN
                      IF (OUTMODE=ABSOLUTE) THEN PC := RELPC;
                      IF (OBJCNT<>0) AND (OUTMODE=ABSOLUTE) THEN FLUSH;
                      OUTMODE := RELATIVE;
                      END;
            ORG  : BEGIN
                      IF (OUTMODE=RELATIVE) THEN RELPC := PC;
                      PC := X.VAL;
                      OUTMODE := ABSOLUTE;
                      IF OBJCNT<>0 THEN FLUSH;
                      END;
            END;
         END;
      IF OBJCNT<>0 THEN FLUSH;
      WRITELN(OBJ,'$');
   END;


(*************************************************************)
(*  BEGIN MAIN BODY OF PROGRAM                               *)
(*************************************************************)

BEGIN
   INIT; HAS := CHR(48); (* INITIALIZE AND DEFINE HAS CHAR (HASH) *)
   LDPRIMFILE;
   IF UNRESOLVED THEN RESOLVE;
   OBJCNT := 0;
   IF NOT UNRESOLVED THEN PASSTWO;
   IF MAP THEN OUTMAP;
END.
END.
```

Z80ASM - Z80 CROSS ASSEMBLER and LINKER USER'S MANUAL

Source Code Listings

Z80ASM -- The Z80 Assembler

```
(*Z80ASM2*)
(*$E+*)
(*$P-,T-*)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*   ZZZZZZZ    88888    00000          AAAAA    SSSSS   M      M   *)
(*        Z    8     8  0     0        A     A  S     S  MM    MM   *)
(*       Z     8     8  0     0        A     A  S        M M  M M   *)
(*      Z       88888   0     0 -----  AAAAAAA   SSSSS   M  M  M   *)
(*     Z       8     8  0     0        A     A        S  M     M   *)
(*    Z        8     8  0     0        A     A  S     S  M     M   *)
(*   ZZZZZZZ    88888    00000         A     A   SSSSS   M     M   *)
(*                                                                       *)
(*   VERSION 1.1                                                         *)
(*   WRITTEN BY GEORGE LEHMANN                                           *)
(*              UNIVERSITY OF ILLINOIS                                   *)
(*              JANUARY, 1978                                            *)
(*                                                                       *)
(*   VERSION 1.2 -- MINOR MODIFICATIONS MADE TO VERSION 1.1              *)
(*   MODIFIED AND MAINTAINED BY RICHARD CONN                             *)
(*                           USA SATELLITE COMMUNICATIONS AGENCY         *)
(*                           MAY, 1979                                   *)
(*                                                                       *)
(*   PROGRAM OPERATION --                                                *)
(*   THE BASIC PROGRAM LOOP IS CONTAINED IN PROC PASS.  AFTER            *)
(*   GETTING A LINE OF TEXT FROM INLINE (READS INPUT) OR MACLINE         *)
(*   (EXPANDS MACROS), THE START-OF-LINE SYMBOL (IF THERE IS ONE)        *)
(*   AND THE INSTRUCTION MNEMONIC ARE PICKED OFF. IF THE INSTRUCTION*)
(*   IS NOT IN THE OPCODE TABLE, GETMAC IS CALLED TO START EXPANSION*)
(*   OF THE MACRO (OR INDICATE AN ERROR IF THE MACRO CAN'T BE FOUND)*)
(*   IF THE INSTRUCTION IS IN THE TABLE, THE NOPNDS VALUE IS USED        *)
(*   TO EVALUATE THE CORRECT NUMBER OF OPERANDS, AND THE SWTCH           *)
(*   VALUE IS USED TO JUMP INTO THE EVALUATION CASE STATEMENT.           *)
(*   THERE IS A SEPERATE CASE FOR EACH INSTRUCTION MNEMONIC.             *)
(*   HERE, OPERAND TYPES ARE CHECKED FOR LEGAL COMBINATIONS AND THE      *)
(*   APPROPRIATE OBJECT CODE IS GENERATED USING VARIOUS EMIT             *)
(*   ROUTINES.                                                           *)
(*   FIVE ADDRESS MODES ARE USED FOR SYMBOLIC VALUES.  RELATIVE          *)
(*   IS FOR WHEN RELOCATABLE CODE (OR DATA) IS BEING GENERATED.          *)
(*   THE VALUE SHOWN IN THE ASSEMBLY LISTING FOR SUCH A LOCATION         *)
(*   IS RELATIVE TO THE BEGINNING OF THE RELOCATABLE ASSEMBLY.           *)
(*   ABSOLUTE IS FOR WHEN CODE (OR DATA) IS PLACED INTO AN ABSOLUTE      *)
(*   LOCATION BY AN ORG INSTRUCTION.  THE VALUE SHOWN WITH THIS          *)
(*   IS THE MEMORY ADDRESS USED BY ALL REFERENCES TO THE LOCATION.       *)
(*   EXTERNAL MODES ARE INDICATED WHEN AND EXT INSTRUCTION IS USED       *)
(*   TO DEFINE A SYMBOL.  IF THE SYMBOL IS ALSO DEFINED ELSEWHERE        *)
(*   IN THE SAME ASSEMBLY, IT BECOMES AND EXTDEFR OR EXTDEFA,            *)
(*   DEPENDING ON THE ADDRESS OF THE SYMBOL BEFORE BEING MADE            *)
(*   EXTERNAL.  IF THE SYMBOL IS NOT DEFINED ELSEWHERE, IT BECOMES       *)
(*   AN EXTREF, ALLOWING RESOLUTION OF ITS ADDRESS TO BE DEFERRED        *)
(*   UNTIL LOAD TIME.                                                    *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
```

```
PROGRAM Z80ASM (SOURCE,LST,OBJ,OUTPUT);

CONST
   OWIDTH = 110;
   IWIDTH = 80;
   OFFSET = 30;
   HEXMAX = 25;
   OBJMAX = 94;
   SMAX   = 10; (* MAXIMUM SYMBOL LENGTH *)
   LMAX   = 56; (* LISTING PAGESIZE      *)
   MSTACK = 50;
   HSKIP  = 2;

TYPE
   WORD = 0..65535;
   SET1 = SET OF 1..48;
   BYTE = 0..255;
   STPTR = ^STE;
   OTPTR = ^OTE;
   SWTYPE = 1..48;
   OPNUM  = 0..2;
   ADRTYPE= (NULL,RELATIVE,ABSOLUTE,EXTDEFA,EXTDEFR,
             EXTREF,REGISTER,IMMEDIATE);
   REFLINK = ^REF;
   REF = RECORD
      ADDR : WORD;
      NEXT : REFLINK
      END;
   STE = PACKED RECORD                (* SYMBOL *)
      NAME : ALFA;                    (* TABLE  *)
      VALUE :WORD;                    (* ENTRY  *)
      PDEF,P2,SPECIAL : BOOLEAN;
      ADMODE : ADRTYPE;
      REFS : REFLINK;
      LSON,RSON : STPTR
      END;
   OTE = RECORD                       (* OPCODE *)
      NAME : ALFA;                    (* TABLE  *)
      DFLT : BYTE;                    (* ENTRY  *)
      SWTCH : SWTYPE;
      NOPNDS : OPNUM;
      LSON,RSON : OTPTR
      END;
   OPNDTYP = (R8,RA,RAF,RBD,RH,RSP,RX,RY,IC,IBD,IH,ISP,
              IX,IY,CF,CF1,CF2,RIR,CON,ICON,RPC);
   OPSET = SET OF OPNDTYP;
   OPERS = (NOOP,PLUS,MINUS,ATIMES,ADIV,AMOD,ANDL,ORL);

VAR
   PASS1,PASS2     : BOOLEAN;
   STFOUND,STHEAD  : STPTR;
   OTFOUND,OTHEAD  : OTPTR;
   OBJ,SOURCE,LST  : TEXT;
   LINECNT,PC,IPOS: INTEGER;
```

```
   HEXLOC,OBJCNT   : INTEGER;
   STRTIME,PGCNT   : INTEGER;
   EPOS,ERRCNT     : INTEGER;
   RELSAVE,CKSUM   : INTEGER;
   SPOS,PDFOUND    : INTEGER;
   PSYM,PINSTR     : ALFA;
   VTIME,VDATE     : ALFA;
   PEXT            : ALFA;
   STOP,REL,INMACRO : BOOLEAN;
   NOTEND,NOTBLNK   : BOOLEAN;
   OPSET1,OPSET2   : OPSET;
   VAL1,VAL2       : WORD;
   DEFLT,DBYTE     : INTEGER;
   ASMTYPE         : ADRTYPE;
   NOXREF,LIST     : BOOLEAN;
   PRGEN,SYMTAB    : BOOLEAN;
   P2XREF,NOSUPRES: BOOLEAN;
   PDVAL : ARRAY[1..31] OF INTEGER;
   PDSET : ARRAY[1..31] OF OPSET;
   OLINE : ARRAY [1..OWIDTH] OF CHAR;
   TITLE,HEADER : ARRAY [1..6] OF ALFA;
   EFLG  : ARRAY [1..13] OF CHAR;
   ASC   : ARRAY [0..63] OF BYTE;
   INP,STR     : ARRAY [1..IWIDTH] OF CHAR;
   SYMBL,INSTR : ARRAY [1..SMAX] OF CHAR;
   PRIORITY    : ARRAY[OPERS] OF INTEGER;
   ONLY8080    : BOOLEAN;
   HASH,SQUOTE,COLON : CHAR;

(*****************************************************************)
(*   INTERNAL DEFINITIONS FOR LOGICAL OPERATION PROCEDURES       *)
(*****************************************************************)

PROCEDURE LAND(A,B:INTEGER; VAR C:INTEGER);
BEGIN
   C := A*B;
END;

PROCEDURE LOR(A,B:INTEGER; VAR C:INTEGER);
BEGIN
   C := A+B;
END;

(*****************************************************************)
(*   THIS PROCEDURE INCREMENTS THE ERROR COUNT AND INDICATES     *)
(*   IT IN THE LISTING.                                          *)
(*****************************************************************)

   PROCEDURE ERROR(INDICATOR : INTEGER);

   BEGIN
      OLINE[3] := HASH;
      ERRCNT := ERRCNT + 1;
      IF EPOS<OFFSET THEN OLINE[EPOS] := EFLG[INDICATOR];
      EPOS := EPOS + 1
```

```
  END;

(*****************************************************************)
(*   THIS FUNCTION TAKES AN INTEGER AS INPUT,                    *)
(*   MODULOS IT TO 0..15 AND RETURNS A CHARACTER FOR ITS         *)
(*   HEXADECIMAL VALUE.                                          *)
(*****************************************************************)

  FUNCTION HEX (I : INTEGER) : CHAR;

  BEGIN
     I := I MOD 16;
     IF I<0 THEN I := I + 16;
     IF I<10 THEN HEX := CHR(ORD('0')+I)
     ELSE HEX := CHR(ORD('A')+I-10)
  END;

(*****************************************************************)
(*   PROCEDURE SRCHST RECURSIVELY SEARCHES THE SYMBOL TABLE.     *)
(*   THE TABLE IS SET UP AS A BINARY TREE, USING LSON AND RSON   *)
(*   AS THE NODE POINTERS.  SRCHST ALWAYS SETS POINTER STFOUND   *)
(*   TO THE LAST ELEMENT WORKED WITH. IF A MATCH WAS FOUND       *)
(*   (SRCHST RETURNS TRUE), STFOUND POINTS TO THE ELEMENT        *)
(*   CORRESPONDING TO THE DESIRED SYMBOL.  IF NO MATCH WAS       *)
(*   FOUND, STFOUND POINTS TO THE ELEMENT TO WHICH THE NEW       *)
(*   NODE SHOULD BE APPENDED.                                    *)
(*****************************************************************)

  FUNCTION SRCHST(SYM : ALFA; PTR : STPTR) : BOOLEAN;

  BEGIN
     STFOUND := PTR; (* POINTER TRAIL*)
     IF SYM = PTR~.NAME THEN SRCHST := TRUE
     ELSE IF SYM < PTR~.NAME THEN
           IF PTR~.LSON = NIL THEN SRCHST := FALSE
           ELSE SRCHST := SRCHST(SYM, PTR~.LSON)
        ELSE IF PTR~.RSON = NIL THEN SRCHST := FALSE
           ELSE SRCHST := SRCHST(SYM,PTR~.RSON)
  END;

(*****************************************************************)
(*   PROCEDURE INSST INSERTS ALL SYMBOLS INTO THE SYMBOL TABLE   *)
(*   NO ERRORS ARE INDICATED DURING PASS 1 IF MULTIPLE           *)
(*   ATTEMPTS ARE MADE TO DEFINE A SYMBOL.  DURING PASS 2, THE   *)
(*   FIRST INSERTION OF A SYMBOL WILL SIMPLY RESULT IN TURNING   *)
(*   ON THE P2 FLAG.  ANY SUBSEQUENT INSERTION ATTEMPT WILL      *)
(*   CAUSE A MULTIPLE-DEFINITION ERROR TO BE INDICATED.          *)
(*   IF A SYMBOL HAS ALREADY BEEN DEFINED BY AN EXT              *)
(*   INSTRUCTION, THE ADDRESS MODE OF THE SYMBOL WILL BE         *)
(*   CHANGED TO EXTDEFR OR EXTDEFA, DEPENDING ON THE INPUT       *)
(*   ADDRESS MODE, ADM.                                          *)
(*****************************************************************)

   PROCEDURE INSST(SYM : ALFA; VAL : INTEGER; PREDEF : BOOLEAN;
             ADM : ADRTYPE; SPCL : BOOLEAN);
```

```
VAR NEWSYM : STPTR;

BEGIN
   IF SRCHST(SYM,STHEAD) THEN
      WITH STFOUND^ DO BEGIN
      IF PASS2 AND P2 THEN ERROR(1);
      IF PASS1 AND (ADMODE=EXTREF) THEN
         BEGIN
         IF ADM=RELATIVE THEN ADMODE := EXTDEFR
         ELSE ADMODE := EXTDEFA;
         VALUE := VAL
         END;
      P2 := (PASS2 AND (ADM<>EXTREF)) OR P2
      END
   ELSE BEGIN
      NEW(NEWSYM);
      WITH NEWSYM^ DO BEGIN
         NAME := SYM;
         VALUE := VAL;
         P2   := FALSE;
         PDEF := PREDEF;
         SPECIAL := SPCL;
         LSON := NIL;
         RSON := NIL;
         REFS := NIL;
         ADMODE := ADM
         END;
      IF SYM<STFOUND^.NAME THEN STFOUND^.LSON := NEWSYM
      ELSE STFOUND^.RSON := NEWSYM
   END
END; (* END PROCEDURE INSST *)

(*****************************************************************)
(*   PROCEDURE ZPAGE SIMPLY PRINTS THE HEADER WHEN REQUESTED.    *)
(*****************************************************************)

   PROCEDURE ZPAGE;

   VAR I : INTEGER;

   BEGIN
      WRITELN(LST);
      IF PGCNT<>1 THEN WRITE(LST,'   PROGRAM ERRORS -- ',ERRCNT:5);
      WRITELN(LST);
      WRITELN(LST,'1');
      LINECNT := 1;
      FOR I := 1 TO HSKIP DO
         BEGIN
         WRITELN(LST);
         LINECNT := LINECNT + 1
         END;
      WRITE(LST,' Z-80 ASSEMBLER  V1.2  ');
      FOR I := 1 TO 6 DO WRITE(LST,TITLE^[I]);
      WRITELN(LST,'      PAGE NUMBER ',PGCNT:5);
```

```
      PGCNT := PGCNT + 1;
      WRITE(LST,'                                    ');
      FOR I:=1 TO 6 DO WRITE(LST,HEADER^[I]);
      WRITELN(LST);
      WRITELN(LST);
      LINECNT := LINECNT + 5
   END;

(*************************************************************)
(*   PROCEDURE DUMPST RECURSIVELY DUMPS SYMBOL TABLE ENTRIES  *)
(*   ACCORDING TO THE FLAG, NOXREF. IF NOXREF IS TRUE, A      *)
(*   COMPACTED SYMBOL TABLE IS PRINTED, GIVING ONLY THE VALUE *)
(*   OF EACH SYMBOL.  IF NOXREF IS FALSE, A LARGER LISTING IS *)
(*   PRODUCED, GIVING MODE, VALUE, AND ALL REFERENCES FOR EACH*)
(*   SYMBOL.  SINCE INORDER TRAVERSAL IS USED, THE SYMBOL     *)
(*   WILL NATURALLY BE SORTED.                                *)
(*************************************************************)

   PROCEDURE DUMPST(PTR : STPTR);

   VAR
      SLINK : REFLINK;

      PROCEDURE WHEX(VAL : WORD);
      BEGIN
         WRITE(LST,' ',HEX(VAL DIV 4096),HEX(VAL DIV 256),
                   HEX(VAL DIV 16),  HEX(VAL))
      END;

      PROCEDURE WRTYPE(TYP : ADRTYPE);
      BEGIN
         CASE TYP OF
            RELATIVE: WRITE(LST,' RELATIVE ');
            ABSOLUTE: WRITE(LST,' ABSOLUTE ');
            EXTDEFA : WRITE(LST,' EXT DEFA ');
            EXTDEFR : WRITE(LST,' EXT DEFR ');
            EXTREF  : WRITE(LST,' EXT REF  ');
            REGISTER: WRITE(LST,' REGISTER ');
            IMMEDIATE:WRITE(LST,' ABSOLUTE ')
            END
      END;

      PROCEDURE SYMPAGE;
      BEGIN
         WRITELN(LST);
         ZPAGE;
         IF NOXREF THEN
            WRITELN(LST,'                          SYMBOL TABLE')
         ELSE WRITELN(LST,' SYMBOL   VALUE ADDR TYPE      REFERENCES');
         WRITELN(LST);WRITELN(LST);
         LINECNT := LINECNT + 3;
         SPOS := 0
      END;

   BEGIN
```

```
      WITH PTR~ DO BEGIN
         IF LSON <> NIL THEN DUMPST(LSON);
         IF LINECNT>LMAX THEN SYMPAGE;
         IF NOT SPECIAL THEN
            IF NOXREF THEN BEGIN
               IF SPOS = 5 THEN BEGIN
                  IF LINECNT>LMAX THEN SYMPAGE;
                  WRITELN(LST);
                  LINECNT := LINECNT+1;
                  SPOS := 0
                  END;
               WRITE(LST,'    ',NAME);
               WHEX (VALUE);
               SPOS := SPOS + 1
               END
            ELSE BEGIN
               SPOS := 0;
               WRITE(LST,' ',NAME);
               WHEX (VALUE);
               WRTYPE(ADMODE);
               SLINK := REFS;
               WHILE SLINK <> NIL DO
                  BEGIN
                  IF SPOS = 15 THEN BEGIN
                     IF LINECNT>LMAX THEN SYMPAGE;
                     WRITELN(LST);
                     WRITE(LST,'                    ');
                     LINECNT := LINECNT+1;
                     SPOS := 0
                     END;
                  IF SPOS= 0 THEN WRITE(LST,'    ');
                  SPOS := SPOS + 1;
                  WHEX(SLINK~.ADDR);
                  SLINK := SLINK~.NEXT
                  END;
               LINECNT := LINECNT + 1;
               WRITELN(LST)
               END;
            IF RSON <> NIL THEN DUMPST(RSON)
         END
      END;

   (*****************************************************************)
   (*   PROCEDURE PDSTE IS MERELY A CONVENIENCE FOR INSERTING       *)
   (*   PREDEFINED ENTRIES INTO THE SYMBOL TABLE.  SINCE ALL        *)
   (*   PREDEFINED ENTRIES USE THEIR VALUE TO REFERENCE ARRAYS      *)
   (*   PDSET AND PDVAL, THIS ROUTINE ALSO BUILDS THESE ARRAYS.     *)
   (*   THESE EXTRA ARRAYS ALLOW THE SYMBOL TABLE ENTRY TO BE       *)
   (*   SMALLER FOR NON-PREDEFINED SYMBOLS, WITHOUT USING VARIANT   *)
   (*   RECORDS.  THESE ARRAYS ALSO CONTAIN EXTRA DEFINITIONS       *)
   (*   (VERY ORDER DEPENDENT!) FOR THE INDIRECT REFERENCES USED    *)
   (*   IN MANY INSTRUCTIONS.  SEE PROC GETOP FOR MORE DETAIL.      *)
   (*****************************************************************)

      PROCEDURE PDSTE(SYM : ALFA; IDX,VAL : INTEGER;
```

```
                    OP : OPSET);

   BEGIN
      INSST(SYM,IDX,TRUE,ABSOLUTE,TRUE);
      PDVAL^[IDX] := VAL;
      PDSET^[IDX] := OP
   END;

(*****************************************************************)
(*   PROCEDURE SRCHOT FUNCTIONS MUCH THE SAME FOR THE OPCODE     *)
(*   TABLE AS SRCHST DOES FOR THE SYMBOL TABLE.                  *)
(*****************************************************************)

   FUNCTION SRCHOT(SYM : ALFA; PTR : OTPTR) : BOOLEAN;

   BEGIN
      OTFOUND := PTR; (* POINTER TRAIL*)
      IF SYM = PTR^.NAME THEN SRCHOT := TRUE
      ELSE IF SYM < PTR^.NAME THEN
            IF PTR^.LSON = NIL THEN SRCHOT := FALSE
            ELSE SRCHOT := SRCHOT(SYM, PTR^.LSON)
         ELSE IF PTR^.RSON = NIL THEN SRCHOT := FALSE
            ELSE SRCHOT := SRCHOT(SYM,PTR^.RSON)
   END;

(*****************************************************************)
(*   PROCEDURE INSOT INSERTS OPCODES INTO THE BINARY-TREE        *)
(*   OPCODE TABLE.  IF A DUPLICITY OCCURS, IT IS A MAJOR ERROR*)
(*   AND THE PROGRAM IS HALTED.                                  *)
(*****************************************************************)

   PROCEDURE INSOT(SYM : ALFA; DEFLT : BYTE;
                   ISWCH : SWTYPE; NOPS : OPNUM);

   VAR NEWSYM : OTPTR;

   BEGIN
      IF SRCHOT(SYM,OTHEAD) THEN HALT;
      TIME(VTIME);
      DATE(VDATE);
      NEW(NEWSYM);
      WITH NEWSYM^ DO BEGIN
         NAME := SYM;
         DFLT := DEFLT;
         SWTCH := ISWCH;
         NOPNDS := NOPS;
         LSON := NIL;
         RSON := NIL
         END;
      IF SYM < OTFOUND^.NAME THEN OTFOUND^.LSON := NEWSYM
      ELSE OTFOUND^.RSON := NEWSYM
   END;

(*****************************************************************)
(*   PROCEDURE INIT IS RESPONSIBLE FOR:                          *)
```

```
(*     1) OPCODE TABLE CONSTRUCTION.                              *)
(*     2) PREDEFINED SYMBOL TABLE CONSTRUCTION.                   *)
(*     3) INITIALIZATION OF ASCII VALUE ARRAY 'ASC'.              *)
(*     4) INITIALIZATION OF ERROR FLAG ARRAY 'EFLG'.              *)
(*     5) INITIALIZATION OF OPERATION PRIORITY ARRAY 'PRIORITY')
(*************************************************************)

   PROCEDURE INIT;

   VAR I : INTEGER;

   BEGIN
      HASH := CHR(48);  (* HASH CHAR *)
      COLON := CHR(0);  (* COLON CHAR *)
      SQUOTE := CHR(56);  (* SINGLE QUOTE CHAR *)
      ONLY8080 := FALSE;
      NEW(STHEAD);
      WITH STHEAD^ DO BEGIN
         NAME := 'I         ';
         VALUE:= 30;
         PDEF := TRUE;
         SPECIAL := TRUE;
         P2    := FALSE;
         LSON := NIL;
         RSON := NIL;
         REFS := NIL
         END;
      PDVAL[30] := 0;
      PDSET[30] := 'RIR';
      PDSTE('C         ',8 ,1 ,'RB,CF,CF1');
      PDSTE('NC        ',24,2 ,'CF,CF1');
      PDSTE('B         ',2 ,0 ,'RB');
      PDSTE('E         ',5 ,3 ,'RB');
      PDSTE('A         ',1 ,7, 'RA,RB');
      PDSTE('AF        ',7 ,3 ,'RAF');
      PDSTE('BC        ',9 ,0 ,'RBD');
      PDSTE('D         ',4 ,2 ,'RB');
      PDSTE('DE        ',10,1 ,'RBD');
      PDSTE('H         ',6 ,4 ,'RB');
      PDSTE('HL        ',11,2 ,'RH');
      PDSTE('IY        ',14,2 ,'RY');
      PDSTE('IX        ',13,2 ,'RX');
      PDSTE('L         ',3 ,5 ,'RB');
      PDSTE('M         ',29,7 ,'CF');
      PDSTE('P         ',28,6 ,'CF');
      PDSTE('NZ        ',22,0 ,'CF,CF1');
      PDSTE('PO        ',26,4 ,'CF');
      PDSTE('PE        ',27,5 ,'CF');
      PDSTE('SP        ',12,3 ,'RSP');
      PDSTE('R         ',31,8 ,'RIR');
      PDSTE('Z         ',23,1 ,'CF,CF1');
      PDVAL[15] := 0;     PDSET[15] := 'IC';
      PDVAL[16] := 0;     PDSET[16] := 'I8D';
      PDVAL[17] := 1;     PDSET[17] := 'I8D';
      PDVAL[18] := 6;     PDSET[18] := 'IH,RB';
```

```
PDVAL^19! := 1;      PDSET^19! := ^ISP!;
PDVAL^20! := 6;      PDSET^20! := ^IX!;
PDVAL^21! := 6;      PDSET^21! := ^IY!;

NEW(OTHEAD);
WITH OTHEAD^ DO BEGIN
   NAME := 'LD        ';
   DFLT := 0;
   SWTCH := 23;
   NOPNDS := 2;
   LSON := NIL;
   RSON := NIL
   END;
INSOT('CALL      ',196,7,1);    INSOT('ADD       ',128,4,2);
INSOT('CCF       ', 63,1,0);    INSOT('CPL       ', 47,1,0);
INSOT('DAA       ', 39,1,0);    INSOT('DI        ',243,1,0);
INSOT('EI        ',251,1,0);    INSOT('EXX       ',217,1,0);
INSOT('HALT      ',118,1,0);    INSOT('NOP       ',  0,1,0);
INSOT('RLA       ', 23,1,0);    INSOT('RLCA      ',  7,1,0);
INSOT('RRA       ', 31,1,0);    INSOT('RRCA      ', 15,1,0);
INSOT('SCF       ', 55,1,0);    INSOT('CPD       ',169,2,0);
INSOT('CPDR      ',185,2,0);    INSOT('CPI       ',161,2,0);
INSOT('CPIR      ',177,2,0);    INSOT('IND       ',170,2,0);
INSOT('INDR      ',186,2,0);    INSOT('INI       ',162,2,0);
INSOT('INIR      ',178,2,0);    INSOT('LDD       ',168,2,0);
INSOT('LDDR      ',184,2,0);    INSOT('LDI       ',160,2,0);
INSOT('LDIR      ',176,2,0);    INSOT('NEG       ', 68,2,0);
INSOT('OUTD      ',171,2,0);    INSOT('OTDR      ',139,2,0);
INSOT('OUTI      ',163,2,0);    INSOT('OTIR      ',179,2,0);
INSOT('RETI      ', 77,2,0);    INSOT('RETN      ', 69,2,0);
INSOT('RLD       ',111,2,0);    INSOT('RRD       ',103,2,0);
INSOT('ADC       ',136,3,2);    INSOT('AND       ',160,5,1);
INSOT('BIT       ', 64,6,2);    INSOT('OR        ',176,5,1);
INSOT('RES       ',128,6,2);    INSOT('SET       ',192,6,2);
INSOT('XOR       ',168,5,1);    INSOT('RET       ',192,8,0);
INSOT('CP        ',184,5,1);    INSOT('JP        ',194,9,1);
INSOT('JR        ', 32,10,1);   INSOT('DEC       ',  5,11,1);
INSOT('INC       ',  4,12,1);   INSOT('EX        ',  0,13,2);
INSOT('IM        ', 70,14,1);   INSOT('OUT       ', 65,15,2);
INSOT('POP       ',193,16,1);   INSOT('PUSH      ',197,16,1);
INSOT('RL        ', 16,17,1);   INSOT('RLC       ',  0,17,1);
INSOT('RR        ', 24,17,1);   INSOT('RRC       ',  8,17,1);
INSOT('SBC       ',152,18,2);   INSOT('SLA       ', 32,17,1);
INSOT('SRA       ', 40,17,1);   INSOT('SRL       ', 56,17,1);
INSOT('SUB       ',144, 5,1);   INSOT('DJNZ      ', 16,20,1);
INSOT('IN        ', 64,21,2);   INSOT('RST       ',199,22,1);
INSOT('EQU       ',0,24,1);     INSOT('ORG       ',0,25,1);
INSOT('REL       ',0,26,0);     INSOT('END       ',0,27,0);
INSOT('DB        ',0,28,0);     INSOT('DW        ',0,29,0);
INSOT('DS        ',0,30,1);     INSOT('EXT       ',0,31,0);
INSOT('TITLE     ',0,32,0);     INSOT('MACRO     ',0,33,0);
INSOT('MEND      ',0,34,0);     INSOT('GEN       ',0,35,0);
INSOT('NOGEN     ',0,36,0);     INSOT('LIST      ',0,37,0);
INSOT('NOLIST    ',0,38,0);     INSOT('XREF      ',0,39,0);
INSOT('NOSYM     ',0,19,0);     INSOT('PAGE      ',0,40,0);
```

```
      INSOT('ONLY8080  ',0,41,0);      INSOT('MESSAGE   ',0,43,0);
      INSOT('HEADER    ',0,42,0);      INSOT('DISPLAY   ',0,44,1);
      INSOT('DEFB      ',0,28,0);      INSOT('DEFM      ',0,28,0);
      INSOT('DEFW      ',0,29,0);      INSOT('DEFS      ',0,30,1);
      INSOT('DUMMY1    ',0,45,0);      INSOT('DUMMY2    ',0,46,0);
      INSOT('DUMMY3    ',0,47,0);      INSOT('DUMMY4    ',0,48,0);
      FOR I := 1 TO 6 DO TITLE[I] := '          ';
      FOR I := 1 TO 6 DO HEADER[I] := '          ';
      ASC[ 0] :=  58;   ASC[37] :=43;
      ASC[38] :=  45;   ASC[39] := 42;
      ASC[40] :=  47;   ASC[41] := 40;
      ASC[42] :=  41;   ASC[43] := 36;
      ASC[44] :=  61;   ASC[45] := 32;
      ASC[46] :=  44;   ASC[47] := 46;
      ASC[48] :=  35;   ASC[49] := 91;
      ASC[50] :=  93;   ASC[51] := 37;
      ASC[52] :=  34;   ASC[53] := 95;
      ASC[54] :=  33;   ASC[55] := 38;
      ASC[56] :=  39;   ASC[57] := 63;
      ASC[58] :=  60;   ASC[59] := 62;
      ASC[60] :=  64;   ASC[61] := 92;
      ASC[62] :=  94;   ASC[63] := 59;
      FOR I := 1 TO 26 DO ASC[I]    := 64+I;
      FOR I := 0 TO 9  DO ASC[27+I] := 48+I;
      EFLG[1] := 'M';   EFLG[2] := 'P';
      EFLG[3] := 'N';   EFLG[4] := 'O';
      EFLG[5] := 'S';   EFLG[6] := 'U';
      EFLG[7] := 'A';   EFLG[8] := 'E';
      EFLG[9] := 'I';   EFLG[10]:= 'F';
      EFLG[11]:= 'R';   EFLG[12]:= 'Q';
      EFLG[13]:= 'Z';
      PRIORITY[NOOP] := 0;     PRIORITY[ORL] := 1;
      PRIORITY[ANDL] := 2;     PRIORITY[PLUS]:= 3;
      PRIORITY[MINUS]:= 3;     PRIORITY[ATIMES] := 4;
      PRIORITY[ADIV] := 4;     PRIORITY[AMOD]:= 4;
   END;

(*****************************************************************)

   PROCEDURE MACLINE;

   BEGIN
      HALT
   END;

(*****************************************************************)

   PROCEDURE GETMAC;

   BEGIN
   ERROR(9)
   END;

(*****************************************************************)
```

```
(*  PROCEDURE EMITCH PUTS THE CHARACTER GIVEN AS A PARAMETER  *)
(*  INTO THE OBJECT FILE AND UPDATES THE CHECKSUM. CHARACTERS *)
(*  0 THRU 9 HAVE VALUES 0 TO 9, AND A THRU Z HAVE VALUES 10  *)
(*  TO 35. ALL OTHER CHARACTERS HAVE VALUE ZERO.              *)
(************************************************************)

PROCEDURE EMITCH (C : CHAR);
BEGIN
   WRITE(OBJ,C);
   IF C IN ['0'..'9'] THEN CKSUM := CKSUM + ORD(C) - ORD('0');
   IF C IN ['A'..'Z'] THEN CKSUM := CKSUM + ORD(C) + 10 - ORD('A');
   OBJCNT := OBJCNT + 1
END;

(************************************************************)
(*   PROCEDURE EMIT PUTS A BYTE INTO THE OBJECT FILE, UPDATES *)
(*   THE CHECKSUM AND INCREMENTS THE PROGRAM COUNTER BY 1.    *)
(************************************************************)

   PROCEDURE EMIT(DATA : INTEGER);
   VAR I : INTEGER;

   BEGIN
      DATA := DATA MOD 256;
      IF DATA < 0 THEN DATA := DATA + 256;
      IF PASS2 THEN BEGIN
         IF HEXLOC > HEXMAX-2 THEN
            BEGIN
            IF LINECNT>LMAX THEN ZPAGE;
            FOR I := 1 TO OWIDTH DO
               BEGIN
               WRITE(LST,OLINE[I]);
               OLINE[I] := ' '
               END;
            WRITELN(LST);
            LINECNT := LINECNT + 1;
            HEXLOC := 11;
            END;
         OLINE[HEXLOC] := HEX(DATA DIV 16);
         OLINE[HEXLOC+1] := HEX(DATA);
         HEXLOC := HEXLOC + 2;
         IF OBJCNT > OBJMAX THEN BEGIN
            OBJCNT := 0;
            WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
            CKSUM := 0
            END;
         IF OBJCNT=0 THEN
            IF ASMTYPE=ABSOLUTE THEN
               BEGIN
               WRITE(OBJ,HASH,HEX(PC DIV 4096),HEX(PC DIV 256),
                     HEX(PC DIV 16),HEX(PC));
               CKSUM := PC + PC DIV 256
               END
            ELSE BEGIN
               WRITE(OBJ,HASH);
```

```
            CKSUM := 0;
            EMITCH('R');
            END;
         OBJCNT := OBJCNT + 2;
         CKSUM := CKSUM + DATA;
         WRITE(OBJ,HEX(DATA DIV 16),HEX(DATA))
         END;
      PC := PC + 1;
   END;

{*************************************************************}
{*   THIS PROC IS RESPNSIBLE FOR GENERATING WORD-VALUED      *}
{*   REFERENCES IN THE OBJECT FILE.  IF THE VALUE IS ABSOLUTE,*}
{*   TWO BYTES ARE EMITTED (LOW-ORDER FIRST).  IF THE VALUE  *}
{*   IS RELATIVE, AN 'R' IS WRITTEN INTO THE OBJECT FILE AND *}
{*   THE PROGRAM LISTING, FOLLOWED BY THE RELATIVE VALUE     *}
{*   (AGAIN, LOW-ORDER BYTE FIRST).   IF THE VALUE IS AN     *}
{*   EXTERNAL REFERENCE, AN 'X' IS WRITTEN INTO THE OBJECT   *}
{*   FILE, FOLLOWED BY THE NAME OF THE REFERENCE, TERMINATED *}
{*   BY A BLANK.  'XXXX' IS WRITTEN INTO THE PROGRAM LISTING. *}
{*************************************************************}

   PROCEDURE EMITW(VALUE : INTEGER; ADM : ADRTYPE);

   VAR SYM : ARRAY[1..SMAXI OF CHAR;
       I : INTEGER;

   BEGIN
      VALUE := VALUE MOD 65536;
      IF VALUE < 0 THEN VALUE := VALUE + 65536;
      HEXLOC := HEXLOC+1;
      IF ADM IN [ABSOLUTE,IMMEDIATE,EXTDEFA] THEN
         BEGIN
         EMIT(VALUE);
         EMIT(VALUE DIV 256);
         END
      ELSE BEGIN
         IF OBJCNT > OBJMAX-10 THEN BEGIN
            OBJCNT := 0;
            WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
            IF ASMTYPE=ABSOLUTE THEN
               BEGIN
               WRITE(OBJ,HASH,HEX(PC DIV 4096),HEX(PC DIV 256),
                     HEX(PC DIV 16),HEX(PC));
               CKSUM := PC + PC DIV 256
               END
            ELSE BEGIN
               WRITE(OBJ,HASH);
               CKSUM := 0;
               EMITCH('R')
               END
            END;
         IF ADM IN [RELATIVE,EXTDEFR] THEN
            BEGIN
            EMITCH('R');
```

```
            OBJCNT := OBJCNT+1;
            OLINE^[HEXLOC] := 'R';
            HEXLOC := HEXLOC+1;
            EMIT(VALUE);
            EMIT(VALUE DIV 256);
            END
         ELSE IF ADM = EXTREF THEN
            BEGIN
            EMITCH('X');
            UNPACK(PEXT^,SYM,1);
            I := 1;
            WHILE (I<=SMAX) AND (SYM^[I]<>' ') DO
               BEGIN
               EMITCH(SYM^[I]);
               I := I + 1;
               END;
            EMITCH(' ');
            IF HEXLOC<HEXMAX-2 THEN
               FOR I := 0 TO 3 DO OLINE^[HEXLOC+I] := 'X';
            HEXLOC := HEXLOC + 6;
            PC := PC + 2
            END
         END
   END;

(*************************************************************)
(*   THIS ROUTINE CHECKS FOR INDIRECT REFERENCES USING THE IX *)
(*   AND IY REGISTERS, AND GENERATES THE PROPER PREFIX IF     *)
(*   SUCH A REFERENCE IS BEING USED.  IT THEN EMITS THE DATA  *)
(*   BYTE, FOLLOWING IT WITH THE VALUE OF THE DISPLACEMENT    *)
(*   INDICATED IN THE ASSEMBLED INSTRUCTION. (THIS VALUE IS   *)
(*   STORED IN GLOBAL VARIABLE 'DBYTE').                      *)
(*************************************************************)

   PROCEDURE EMITXY(DATA : INTEGER);

   BEGIN
      IF (IX IN OPSET1) OR (IX IN OPSET2) THEN EMIT(221)
      ELSE  IF (IY IN OPSET1) OR (IY IN OPSET2) THEN EMIT(253);
      EMIT(DATA);
      IF (^IX,IY^*OPSET1<>^[]) OR (^IX,IY^*OPSET2<>^[]) THEN
         BEGIN
         IF ONLY8080 THEN ERROR(13);
         EMIT(DBYTE)
         END
   END;

(*************************************************************)
(*   THIS ROUTINE MERELY EMITS 'ED' HEXADECIMAL FOLLOWED BY  *)
(*   THE INPUT DATA VALUE.  THIS IS ONLY A CONVENIENCE.      *)
(*************************************************************)

   PROCEDURE EMITED (DATA : INTEGER);

   BEGIN
```

```
      IF ONLY8080 THEN ERROR(13);
      EMIT(237);
      EMIT(DATA)
   END;

(*****************************************************************)
(*   THIS ROUTINE CHECKS FOR USE OF IX AND IY IN A NON-INDIRECT  *)
(*   FASHION.  IF THIS IS FOUND, THE PROPER PREFIX IS GENERATED  *)
(*   DBYTE IS NOT EMITTED.                                       *)
(*****************************************************************)

   PROCEDURE EMITXY0 (DATA : INTEGER);

   BEGIN
      IF (RX IN OPSET1) OR (RX IN OPSET2) THEN
         BEGIN
         IF ONLY8080 THEN ERROR(13);
         EMIT(221);
         END
      ELSE IF (RY IN OPSET1) OR (RY IN OPSET2) THEN
         BEGIN
         IF ONLY8080 THEN ERROR(13);
         EMIT(253);
         END;
      EMIT(DATA)
   END;

(*****************************************************************)
(*   PROCEDURE OUTESD RECURSIVELY SEARCHES THE SYMBOL TABLE      *)
(*   FOR ALL EXTERNAL DEFINITIONS AND REFERENCES.  IT THEN       *)
(*   WRITES THESE REFERENCES INTO THE OBJECT FILE IN AN 'S'      *)
(*   BLOCK(S).                                                   *)
(*****************************************************************)

   PROCEDURE OUTESD (PTR : STPTR);

   PROCEDURE OUTX(N : ALFA; A : ADRTYPE; V : INTEGER);

   VAR C : CHAR;
      NCHAR : ARRAY[1..10] OF CHAR;
      I     : INTEGER;
   BEGIN
      IF OBJCNT>OBJMAX-16 THEN
         BEGIN
         WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
         OBJCNT := 0
         END;
      IF OBJCNT=0 THEN
         BEGIN
         WRITE(OBJ,HASH);
         CKSUM := 0;
         EMITCH('S')
         END;
      CASE A OF
         EXTREF : C := 'F';
```

```
            EXTDEFA: C := 'A';
            EXTDEFR: C := 'R'
            END;
         EMITCH(C);
         UNPACK(N,NCHAR,1);
         I := 1;
         WHILE (I<=SMAX) AND (NCHAR^[I]<>' ') DO
            BEGIN
            EMITCH(NCHAR^[I]);
            I := I + 1
            END;
         EMITCH(' ');
         IF (A=EXTDEFA) OR (A=EXTDEFR) THEN EMITW(V,ABSOLUTE);
      END;

      BEGIN
         WITH PTR~ DO BEGIN
            IF LSON <> NIL THEN OUTESD(LSON);
            IF ADMODE IN [EXTDEFA,EXTDEFR,EXTREF] THEN
               OUTX(NAME,ADMODE,VALUE);
            IF RSON <> NIL THEN OUTESD(RSON)
            END
      END;
   (*****************************************************************)
   (*   THIS PROCEDURE WRITES THE CURRENT PC INTO THE LISTING.      *)
   (*****************************************************************)

      PROCEDURE ADDRESS;

      BEGIN
         OLINE^[HEXLOC]   := HEX(PC DIV 4096);
         OLINE^[HEXLOC+1] := HEX(PC DIV 256);
         OLINE^[HEXLOC+2] := HEX(PC DIV 16);
         OLINE^[HEXLOC+3] := HEX(PC);
         HEXLOC := HEXLOC + 6
      END;

   (*****************************************************************)
   (*   PROCEDURE INLINE READS A LINE FROM THE INPUT FILE AND       *)
   (*   PUTS IT IN GLOBAL BUFFER ARRAY INP.  IF NO NON-BLANK        *)
   (*   CHARACTERS WERE ENCOUNTERED, GLOBAL BOOLEAN NOTBLANK IS     *)
   (*   FALSE AT EXIT.                                              *)
   (*****************************************************************)

      PROCEDURE INLINE;

      VAR  I : INTEGER;

      BEGIN
         EPOS := HEXMAX+1;
         FOR I := 1 TO IWIDTH DO INP^[I] := ' ';
         I := 0;
         STOP := EOF(SOURCE);
         NOTBLNK := FALSE;
         WHILE (I<IWIDTH) AND NOT STOP DO
```

```
        BEGIN
        I := I + 1;
        IF EOLN(SOURCE) THEN
           BEGIN
           READLN(SOURCE);
           I := IWIDTH + 1
           END
        ELSE BEGIN
           READ(SOURCE,INP^[I]);
           IF INP^[I]<>' ' THEN NOTBLNK := TRUE;
           IF (I=IWIDTH) THEN READLN(SOURCE)
           END
        END
   END;

(*****************************************************************)
(*   THIS PROCEDURE MOVES THE INPUT POINTER (IPOS) UNTIL A       *)
(*   NON-BLANK CHARACTER IS ENCOUNTERED, OR THE END IS REACHED*)
(*****************************************************************)

   PROCEDURE SKPBLNK;

   BEGIN
      WHILE (INP^[IPOS]= ' ') AND (IPOS < IWIDTH) DO IPOS := IPOS + 1
   END;

(*****************************************************************)
(*   PROCEDURE GETOP EVALUATES ALL OPERANDS EXCEPT STRINGS       *)
(*   ENCOUNTERED IN TITLE AND DB OPS.  IF THE OPERAND IS         *)
(*   ENCLOSED IN PARENTHESES, AN INDIRECT REFERENCE IS           *)
(*   INDICATED.                                                  *)
(*   INTERNAL PROCEDURES EVAL1 AND EXPR ARE USED IN EVALUATING*)
(*   THE OPERAND.                                                *)
(*****************************************************************)

   PROCEDURE GETOP (VAR OPS : OPSET; VAR VAL : WORD;
                    VAR ADM : ADRTYPE);

   VAR
      INDRCT : BOOLEAN;

      FUNCTION HEXVAL(CH : CHAR) : INTEGER;

      BEGIN
         IF CH IN ['0'..'9'] THEN HEXVAL := ORD(CH) - ORD('0')
         ELSE IF CH IN ['A'..'F'] THEN HEXVAL := ORD(CH) + 10 - ORD('A')
            ELSE BEGIN
               HEXVAL := 0;
               ERROR(3)
               END
      END;

   (*************************************************************)
   (*   EVAL1 EVALUATES ONE FACTOR AT A TIME.  IT DOES THE      *)
   (*   SYMBOL TABLE LOOKUP, CHARACTER-TO-INTEGER CONVERSIONS *)
```

```
(*   AND SPECIAL ITEMS $ AND SINGLE-BYTE CHARACTERS.  IF    *)
(*   CROSS-REFERENCING OF THE SYMBOLS IS DESIRED, THE LISTS*)
(*   ARE BUILT HERE DURING PASS 2 (CONTROLLED BY BOOLEAN    *)
(*   VARIABLE P2XREF                                        *)
(*********************************************************)
   PROCEDURE EVAL1(VAR VAL : INTEGER; VAR OPS : OPSET;
                   VAR ADM : ADRTYPE);

   VAR
      SYM    : ARRAY[1..SMAX] OF CHAR;
      PSYM   : ALFA;
      I,J    : INTEGER;
      UNARYM : BOOLEAN;

      PROCEDURE REFLIST;
      VAR
         NOMATCH : BOOLEAN;
         REFL,LASTREF,NEWREF : REFLINK;
      BEGIN
      WITH STFOUND^ DO
         IF REFS=NIL THEN
            BEGIN
            NEW(NEWREF);
            NEWREF^.ADDR := PC;
            NEWREF^.NEXT := NIL;
            REFS := NEWREF
            END
         ELSE BEGIN
            REFL := REFS;
            NOMATCH := TRUE;
            WHILE (REFL<>NIL) AND NOMATCH DO
               BEGIN
               NOMATCH := REFL^.ADDR<>PC;
               LASTREF := REFL;
               REFL := REFL^.NEXT
               END;
            IF NOMATCH THEN
               BEGIN
               NEW(NEWREF);
               NEWREF^.ADDR := PC;
               NEWREF^.NEXT := NIL;
               LASTREF^.NEXT:= NEWREF
               END
            END
      END;

   BEGIN
      OPS := [];
      ADM := NULL;
      PDFOUND := 0;
      UNARYM := FALSE;
      IF INP[IPOS] = '(' THEN
         BEGIN
         INDRCT := TRUE;
         IPOS := IPOS + 1
```

```
   END;
IF INP^IPOS!='-' THEN
   BEGIN
   UNARYM := TRUE;
   IPOS := IPOS + 1
   END;
FOR I := 1 TO SMAX DO SYM^I! := ' ';
I := 0;
WHILE (I<SMAX) AND (IPOS<=IWIDTH) AND
   NOT (INP^IPOS! IN [' ',',',')','+','-','*','/','%','&','|']) DO
   BEGIN
   I := I +1;
   SYM^I! := INP^IPOS!;
   IPOS := IPOS +1
   END;
IF SYM^1! IN ['A'..'Z'] THEN
   BEGIN
   IF SYM^3!=SQUOTE THEN SYM^3!:= ' ';
   PACK(SYM,1,PSYM);
   IF SRCHST(PSYM,STHEAD) THEN
      WITH STFOUND^ DO
         IF PDEF THEN BEGIN
            IF INDRCT THEN PDFOUND := VALUE+7
            ELSE PDFOUND := VALUE;
         IF (NOT SPECIAL AND P2XREF) THEN REFLIST;
         OPS := POSET^PDFOUND!;
         VAL := PDVAL^PDFOUND!;
         ADM := REGISTER
         END
      ELSE BEGIN
         IF P2XREF THEN REFLIST;
         IF ((PINSTR='EQU       ')OR(PINSTR='DS        '))
            AND NOT P2 THEN ERROR(10);
         PEXT := PSYM;
         VAL := VALUE;
         OPS := ^CON!;
         ADM := ADMODE
         END
   ELSE BEGIN
      ERROR(6);
      OPS := ^CON!;
      VAL := 0;
      ADM := ABSOLUTE
      END
   END
ELSE IF SYM^1! IN ['0'..'9'] THEN
   BEGIN
   IF SYM^I! = 'H' THEN
      BEGIN
      VAL := 0;
      FOR J := 1 TO I-1 DO VAL := VAL*16 + HEXVAL(SYM^J!)
      END
   ELSE BEGIN
      VAL := 0;
      FOR J := 1 TO I DO
```

```
            BEGIN
            VAL := VAL*10 + ORD(SYM[J])-27;
            IF NOT (SYM[J] IN ['0'..'9']) THEN ERROR(3)
            END
         END;
      OPS := [CON];
      ADM := IMMEDIATE
      END
   ELSE IF SYM[1]=SQUOTE THEN
      BEGIN
      IF (I=1) THEN
         BEGIN
         SYM[2] := INP[IPOS];
         IPOS := IPOS + 2
         END;
      VAL := ASC[ORD(SYM[2])];
      OPS := [CON];
      ADM := IMMEDIATE
      END
   ELSE IF SYM[1] = '$' THEN
      BEGIN
      VAL := PC;
      OPS := [RPC];
      ADM := ASMTYPE
      END
   ELSE ERROR(4);
   IF UNARYM THEN VAL := -VAL;
   VAL := VAL MOD 65536;
   IF VAL<0 THEN VAL := VAL + 65536;
END;

(*************************************************************)
(*   THIS FUNCTION EVALUATES EXPRESSIONS IN THE OPERAND      *)
(*   FIELD.  SINCE PARENTHESES INDICATE INDIRECTION, THEY    *)
(*   ARE NOT ALLOWED IN EXPRESSIONS.  THE OPERATORS IN       *)
(*   DECREASING ORDER OF PRECEDENCE ARE: (ROWS ARE EQUAL)    *)
(*    MODULO('%'), DIVISION('/'), MULTIPLICATION('*')        *)
(*    PLUS('+'), MINUS('-')                                  *)
(*    AND('&'), OR('!')                                      *)
(*   ADDRESS MODES ARE NOT PERMITTED TO BE MIXED             *)
(*   INDISCRIMINATELY.  RELATIVE AND EXTDEFR MODES CAN ONLY  *)
(*   BE OPERATED ON BY PLUS AND MINUS.  NO OPERATIONS ARE    *)
(*   ALLOWED ON EXTREF'S.  *)
(*************************************************************)
   FUNCTION EXPR (VAL0 : INTEGER; VAR ADM : ADRTYPE) : INTEGER;

   VAR
      VSTACK,OSTACK : INTEGER;
      VALUES        : ARRAY[1..MSTACK] OF INTEGER;
      OPERATORS     : ARRAY[1..MSTACK] OF OPERS;
      ADMODES       : ARRAY[1..MSTACK] OF ADRTYPE;
      TVAL : INTEGER;
      TOPS : OPSET;
      TADM : ADRTYPE;
```

```
PROCEDURE PUSHV(VAL : INTEGER; ADM : ADRTYPE);
BEGIN
   VSTACK := VSTACK + 1;
   VALUES[VSTACK] := VAL;
   ADMODES[VSTACK] := ADM
END;

FUNCTION POPV (VAR ADM : ADRTYPE) : INTEGER;
BEGIN
   POPV := VALUES[VSTACK];
   ADM  := ADMODES[VSTACK];
   VSTACK := VSTACK-1
END;

FUNCTION OPCHAR(CH : CHAR) : OPERS;
BEGIN
   CASE CH OF
   '+': OPCHAR := PLUS;
   '-': OPCHAR := MINUS;
   '*': OPCHAR := ATIMES;
   '/': OPCHAR := ADIV;
   '%': OPCHAR := AMOD;
   '&': OPCHAR := ANDL;
   '|': OPCHAR := ORL
   END
END;

PROCEDURE PUSHO (VAL : OPERS);
BEGIN
   OSTACK := OSTACK + 1;
   OPERATORS[OSTACK] := VAL
END;

FUNCTION POPO : OPERS;
BEGIN
   POPO := OPERATORS[OSTACK];
   OSTACK:= OSTACK - 1
END;

FUNCTION TOPOP : OPERS;
BEGIN
   TOPOP := OPERATORS[OSTACK]
END;

PROCEDURE PERFORM;
VAR V1,V2,V3 : INTEGER;
    A1,A2 : ADRTYPE;
    OP    : OPERS;
BEGIN
   V2 := POPV(A2);
   V1 := POPV(A1);
   OP := POPO;
   CASE OP OF
   NOOP : HALT;
   PLUS : V1 := V1 + V2;
```

```
        MINUS: V1 := V1 - V2;
        ATIMES:V1 := V1 * V2;
        ADIV : V1 := V1 DIV V2;
        AMOD : V1 := V1 MOD V2;
        ANDL : BEGIN LAND(V1,V2,V3); V1 := V3; END;
        ORL  : BEGIN LOR(V1,V2,V3); V1 := V3; END;
        END;
        IF ('REGISTER,EXTREF!*'A1,A2!<>'!) OR
           ((RELATIVE IN 'A1,A2!) AND (OP IN 'ATIMES,ADIV,AMOD,ANDL,ORL!))
           THEN BEGIN
              ERROR(7);
              A1 := NULL
              END
        ELSE IF RELATIVE IN 'A1,A2! THEN A1 := RELATIVE
        ELSE IF ABSOLUTE IN 'A1,A2! THEN A1 := ABSOLUTE
        ELSE IF EXTDEFR IN 'A1,A2! THEN A1 := EXTDEFR
        ELSE IF EXTDEFA IN 'A1,A2! THEN A1 := EXTDEFA
        ELSE IF IMMEDIATE IN 'A1,A2! THEN A1 := IMMEDIATE
        ELSE ERROR(8);
        PUSHV(V1,A1)
     END;


  BEGIN
     VSTACK := 0;
     OSTACK := 0;
     PUSHV (VAL0,ADM);
     PUSHO (NOOP);
     WHILE INP'IPOS! IN ''+','-','*','/','%','&','|'! DO
        BEGIN
        WHILE PRIORITY'OPCHAR(INP'IPOS!)!<=PRIORITY'TOPOP! DO
           PERFORM;
        PUSHO(OPCHAR(INP'IPOS!));
        IPOS := IPOS + 1;
        EVAL1(TVAL,TOPS,TADM);
        PUSHV(TVAL,TADM)
        END;
     WHILE PRIORITY'TOPOP!>0 DO PERFORM;
     TVAL := POPV(ADM) MOD 65536;
     IF TVAL<0 THEN TVAL:= TVAL+65536;
     EXPR := TVAL
  END;

BEGIN
  SKPBLNK;
  INDRCT := FALSE;
  IF INP'IPOS!=',' THEN IPOS := IPOS + 1;
  EVAL1(VAL,OPS,ADM);
  IF 'IX,IY!*OPS<>'! THEN
     BEGIN
        ADM := RELATIVE;
        DBYTE := EXPR(0,ADM);
        ADM := IMMEDIATE
     END
  ELSE VAL := EXPR(VAL,ADM);
```

```
      IF (CON IN OPS) AND INDRCT THEN OPS := 'ICON!;
      IF INDRCT THEN IPOS := IPOS + 1
   END;

(*************************************************************)
(*   PROCEDURE PASS IS THE HEART OF THE ASSEMBLER.  IT CONTAINS)
(*   THE BASIC LOOP FOR SCANNING THE INPUT AND CO-ORDINATING  *)
(*   THE OTHER PROCEDURES.  THE FIRST OPERATION IS TO GET A LINE
(*   OF TEXT.  IF BOOLEAN VARIABLE INMACRO IS TRUE, THE LINE IS)
(*   GENERATED BY PROCEDURE MACLINE (IMPLEMENTED ONLY AS AN ERROR
(*   CALL IN V1.0).  THE BEGINNING-OF-LINE SYMBOL(IF ANY) IS  *)
(*  COLLECTED INTO VARIABLE PSYM, AND THE INSTRUCTION         *)
(*   MNEMONIC IS PUT INTO VARIABLE PINSTR.  IF THE INSTRUCTION*)
(*   IS FOUND IN THE OPCODE TABLE, PROCESSING BEGINS.  IF NOT,*)
(*   THE SYMBOL IS PASSED TO PROCEDURE GETMAC, WHICH WILL LOOK*)
(*   THE SYMBOL UP IN A MACRO TABLE. (GETMAC IS ONLY AN ERROR *)
(*   CALL IN V1.0)  FIELD NOPNDS OF THE FOUND OPCODE TABLE    *)
(*   ELEMENT TELLS HOW MANY OPERANDS TO EVALUATE BEFORE STARTING
(*   THE INSTRUCTION-PROCESSING CASE STATEMENT.  FIELD SWTCH  *)
(*   IS THE VALUE USED TO JUMP INTO THE CASE STATEMENT.       *)
(*   THE CASE STATEMENT ITSELF IS MERELY A COLLECTION OF TESTS*)
(*   TO SEE IF THE OPERANDS FALL INTO CERTAIN CLASSES FOR THE *)
(*   PARTICULAR INSTRUCTION.  CODE SEQUENCES ARE THEN GENERATED)
(*   WHEN AN ACCEPTABLE COMBINATION OF OPERANDS IS FOUND.     *)
(*   THE LOOP IS ENDED WITH THE PRINT STATEMENT WHICH IS      *)
(*   EXECUTED ONLY IF IT IS PASS2 AND THE NOT-SUPPRESS FLAG IS*)
(*   TRUE.  THUS SPAKE ZARATHUSTRA                            *)
(*************************************************************)

   PROCEDURE PASS (PARM : INTEGER);

   VAR
   I,J : INTEGER;
   ADM1,ADM2 : ADRTYPE;

      FUNCTION STRING : INTEGER;

      VAR   I : INTEGER;
         EXIT : BOOLEAN;

      BEGIN
         I := 0;
         IPOS := IPOS + 1;
         EXIT := FALSE;
         REPEAT
            IF IPOS > IWIDTH THEN
               BEGIN
               EXIT := TRUE;
               ERROR(5)
               END
            ELSE IF INP'IPOS! <> SQUOTE THEN
               BEGIN
               I := I + 1;
               STR'I! := INP'IPOS!;
               IPOS := IPOS + 1
```

```
            END
         ELSE IF (IPOS<IWIDTH) AND (INP^IPOS+1!=SQUOTE) THEN
            BEGIN
            I := I + 1;
            STR^I! := SQUOTE;
            IPOS := IPOS + 2
            END
         ELSE EXIT := TRUE
      UNTIL EXIT;
      IPOS := IPOS + 1;
      STRING := I
   END;

   PROCEDURE INCDEC(DEFLT2 : BYTE);

   BEGIN
      IF R8 IN OPSET1 THEN EMIT(DEFLT + VAL1*8)
      ELSE IF ^R8D,RH,RSP!*OPSET1<>^! THEN EMIT(DEFLT2 + VAL1*16)
      ELSE IF ^IX,IY!*OPSET1<>^! THEN EMITXY(DEFLT+48)
      ELSE IF ^RX,RY!*OPSET1<>^! THEN EMITXY0(DEFLT2 + 32)
      ELSE ERROR(4)
   END;

BEGIN
   PASS1 := PARM = 1;
   PASS2 := PARM = 2;
   P2XREF:= PASS2 AND NOT NOXREF;
   LINECNT := 100;
   INMACRO := FALSE;
   STOP    := FALSE;
   NOTEND  := TRUE;
   ASMTYPE := ABSOLUTE;
   RELSAVE := 0;
   NOXREF  := TRUE;
   LIST    := TRUE;
   SYMTAB := TRUE;
   PC      := 0;
   ERRCNT  := 0;
   PGCNT   := 1;
   RESET(SOURCE);
   REPEAT
      IF INMACRO THEN MACLINE
      ELSE INLINE;
      DBYTE := 0;
      NOSUPRES := TRUE;
      FOR I := 1 TO OFFSET DO OLINE^I! := ' ';
      OLINE^OFFSET! := COLON;
      FOR I := 1 TO IWIDTH DO
         OLINE^I+OFFSET! := INP^I!;
      FOR I := 1 TO SMAX DO SYMBL^I! := ' ';
      IPOS := 1;
      IF ((INP^1! <> '*') AND (INP^1!<> ';'))  AND NOTBLNK THEN
         BEGIN                        (* START LINE PROCESSING*)
         WHILE (IPOS<SMAX+1) AND (INP^IPOS! <> ' ') DO
            BEGIN                     (* COLLECT SYMBOL *)
```

```
   SYMBL^.IPOS! := INP^.IPOS!;(* AT START OF    *)
   IPOS := IPOS + 1          (* LINE, IF ANY   *)
   END;
PACK (SYMBL,1,PSYM);
(* NOW SKIP REST OF LONG SYMBOL *)
WHILE INP^.IPOS! <> ' ' DO IPOS := IPOS + 1;
SKPBLNK;
FOR I := 1 TO SMAX DO INSTR^.I! := ' ';
I := IPOS-1;
WHILE INP^.IPOS! <> ' ' DO
   BEGIN
   INSTR^.IPOS-I! := INP^.IPOS!;
   IPOS := IPOS + 1
   END;
PACK(INSTR,1,PINSTR);
IF NOT SRCHOT(PINSTR,OTHEAD) THEN GETMAC
ELSE BEGIN
   DEFLT := OTFOUND^.DFLT;
   IF OTFOUND^.NOPNDS > 0 THEN GETOP(OPSET1,VAL1,ADM1);
   IF OTFOUND^.NOPNDS = 2 THEN GETOP(OPSET2,VAL2,ADM2);
   HEXLOC := 5;
   IF OTFOUND^.SWTCH IN ^1..23,28..30! THEN
      BEGIN
      ADDRESS:
      IF PSYM<>'          ' THEN INSST(PSYM,PC,FALSE,ASMTYPE,FALSE)
      END;
   CASE OTFOUND^.SWTCH OF

      (* 1-BYTE, NO-OPERAND INSTRUCTIONS *)

      1:EMIT(DEFLT);

      (* 2-BYTE, NO-OPERAND INSTRUCTIONS *)

      2: EMITED(DEFLT);

      (* ADC INSTRUCTION *)

      3: IF RA IN OPSET1 THEN
            IF RB IN OPSET2 THEN
               EMIT(136+VAL2)
            ELSE IF (^IX,IY!*OPSET2<>^!) THEN
               EMITXY(142)
            ELSE IF CON IN OPSET2 THEN
               BEGIN
               EMIT(206);
               EMIT(VAL2)
               END
            ELSE ERROR(4)
         ELSE IF (RH IN OPSET1) AND (^RBD,RH,RSP!*OPSET2<>^!) THEN
            EMITED(74 + VAL2*16)
         ELSE ERROR(4);

      (* ADD INSTRUCTION *)
```

```
4: IF RA IN OPSET1 THEN
      IF R8 IN OPSET2 THEN
         EMIT(128+VAL2)
      ELSE IF 'IX,IY!*OPSET2<>'! THEN
         EMITXY(134)
      ELSE IF CON IN OPSET2 THEN
         BEGIN
         EMIT(198);
         EMIT(VAL2)
         END
      ELSE ERROR(4)
   ELSE IF (RH IN OPSET1) AND ('RBD,RH,RSP!*OPSET2<>'!) THEN
      EMIT(9 + VAL2*16)
   ELSE IF('RX,RY!*OPSET1<>'!)AND('RBD,RX,RY,RSP!*OPSET2<>'!)
      THEN EMITXYO(9 + VAL2*16)
   ELSE ERROR(4);

(* AND,OR,XOR,CP, & SUB INSTRUCTIONS *)

5: IF R8 IN OPSET1 THEN
      EMIT(DEFLT + VAL1)
   ELSE IF 'IX,IY!*OPSET1<>'! THEN
      EMITXY(DEFLT + 6)
   ELSE IF CON IN OPSET1 THEN
      BEGIN
      EMIT(DEFLT + 70);
      EMIT(VAL1)
      END
   ELSE ERROR(4);

(* BIT,SET, & RES INSTRUCTIONS *)

6: BEGIN
   IF ONLY8080 THEN ERROR(13);
   IF CON IN OPSET1 THEN
      IF R8 IN OPSET2 THEN
         BEGIN
         EMIT(203);
         EMIT(DEFLT + VAL2 + VAL1*8)
         END
      ELSE IF 'IX,IY!*OPSET2<>'! THEN
         BEGIN
         EMITXY(203);
         EMIT(DEFLT + 6 + VAL1*8)
         END
      ELSE ERROR(4)
   ELSE ERROR(4);
END;

(* CALL INSTRUCTION *)

7: IF CON IN OPSET1 THEN
      BEGIN
      EMIT(205);
      EMITW(VAL1,ADM1)
```

```
      END
    ELSE IF CF IN OPSET1 THEN
      BEGIN
      IF R8 IN OPSET1 THEN VAL1 := 3;
      GETOP(OPSET2,VAL2,ADM2);
      IF CON IN OPSET2 THEN
         BEGIN
         EMIT(196 + VAL1*8);
         EMITW(VAL2,ADM2)
         END
      ELSE ERROR(4)
      END
    ELSE ERROR(4);

(* RET INSTRUCTION *)

8: BEGIN
   SKPBLNK;
   OPSET1 := [];
   IF IPOS<IWIDTH THEN
      BEGIN
      FOR I := 1 TO SMAX DO SYMBL[I] := ' ';
      I := 1;
      WHILE ((I<4)AND(IPOS<IWIDTH))AND(INP[IPOS]<>' ') DO
         BEGIN
         SYMBL[I] := INP[IPOS];
         I := I+1;
         IPOS := IPOS+1;
         END;
      PACK(SYMBL,1,PSYM);
      IF SRCHST(PSYM,STHEAD) THEN
         WITH STFOUND^ DO
         IF PDEF THEN
            BEGIN
            OPSET1 := PDSET[VALUE];
            VAL1   := PDVAL[VALUE]
            END
      END;
    IF OPSET1=[] THEN
      EMIT(201)
    ELSE IF CF IN OPSET1 THEN
      BEGIN
      IF R8 IN OPSET1 THEN VAL1 := 3;
      EMIT(192 + VAL1*8)
      END
    ELSE ERROR(4)
    END;

 (* JP INSTRUCTION *)

 9: IF CON IN OPSET1 THEN
       BEGIN
       EMIT(195);
       EMITW(VAL1,ADM1)
       END
```

```
   ELSE IF CF IN OPSET1 THEN
      BEGIN
      IF R8 IN OPSET1 THEN VAL1 := 3;
      GETOP(OPSET2,VAL2,ADM2);
      EMIT(194 + VAL1*8);
      EMITW(VAL2,ADM2)
      END
   ELSE IF [IH,IX,IY]*OPSET1<>[] THEN
      BEGIN
      IF IY IN OPSET1 THEN
         BEGIN
         IF ONLY8080 THEN ERROR(13);
         EMIT(253)
         END
      ELSE IF IX IN OPSET1 THEN
         BEGIN
         IF ONLY8080 THEN ERROR(13);
         EMIT(221)
         END;
      EMIT(233)
      END
   ELSE ERROR(4);

(* JR INSTRUCTION *)

10:BEGIN
   IF ONLY8080 THEN ERROR(13);
   IF CON IN OPSET1 THEN
      BEGIN
      EMIT(24);
      EMIT(VAL1-2);
      IF (VAL1<2) OR ((VAL1 >129) AND (VAL1<65410))
         THEN ERROR(11)
      END
   ELSE IF CF1 IN OPSET1 THEN
      BEGIN
      IF R8 IN OPSET1 THEN VAL1 := 3;
      GETOP(OPSET2,VAL2,ADM2);
      EMIT(32 + VAL1*8);
      EMIT(VAL2-2);
      IF (VAL2<2) OR ((VAL2 >129) AND (VAL2<65410))
         THEN ERROR(11)
      END
   ELSE ERROR(4);
END;

(* DEC INSTRUCTION *)

11:INCDEC(11);

(* INC INSTRUCTION *)

12:INCDEC(3);

(* EX INSTRUCTION *)
```

```
13:IF (ISP IN OPSET1) AND ('RH,RX,RY!*OPSET2<>'!) THEN
      EMITXY0(227)
   ELSE IF (RAF IN OPSET1) AND (RAF IN OPSET2) THEN
      BEGIN
      IF ONLY8080 THEN ERROR(13);
      EMIT(8);
      END
   ELSE IF (RBD IN OPSET1) AND (RH IN OPSET2) THEN EMIT(235)
   ELSE ERROR(4);

(* IM INSTRUCTION *)

14:IF (VAL1<0) OR (VAL1>2) THEN ERROR(4)
   ELSE CASE VAL1 OF
      0: EMITED(70);
      1: EMITED(86);
      2: EMITED(94)
      END;

(* OUT INSTRUCTION *)

15:IF (IC IN OPSET1) AND (R8 IN OPSET2) THEN
      EMITED(65 + VAL2*8)
   ELSE IF (ICON IN OPSET1) AND (RA IN OPSET2) THEN
      BEGIN
      EMIT(211);
      EMIT(VAL1)
      END
   ELSE ERROR(4);

(* PUSH & POP INSTRUCTIONS *)

16:IF 'RAF,RBD,RH,RX,RY!*OPSET1<>'! THEN
      EMITXY0(DEFLT + VAL1*16)
   ELSE ERROR(4);

(* RL,RLC,RR,RRC,SLA,SRA, & SRL INSTRUCTIONS *)

17:BEGIN
   IF ONLY8080 THEN ERROR(13);
   IF 'R8,RA,IH!*OPSET1<>'! THEN EMIT(203)
   ELSE IF 'IX,IY!*OPSET1<>'! THEN EMITXY(203)
   ELSE ERROR(4);
   EMIT(DEFLT + VAL1)
   END;

(* SBC INSTRUCTION *)

18:IF RA IN OPSET1 THEN
      IF R8 IN OPSET2 THEN EMIT(152 + VAL2)
      ELSE IF 'IX,IY!*OPSET2<>'! THEN EMITXY(152)
      ELSE IF CON IN OPSET2 THEN
         BEGIN
         EMIT(222);
```

```
        EMIT(VAL2)
        END
     ELSE ERROR(4)
  ELSE IF (RH IN OPSET1) AND
     ('RBD,RH,RSP!*OPSET2<>'!) THEN
     EMITED(66 + VAL2*16)
  ELSE ERROR(4);

(* NOSYM PSEUDO-OP *)

19:SYMTAB := FALSE;

(* DJNZ INSTRUCTION *)

20:BEGIN
   EMIT(16);
   EMIT(VAL1-2)
   END;

(* IN INSTRUCTION *)

21:IF (R8 IN OPSET1) AND (IC IN OPSET2) THEN
      EMITED(64 + VAL1*8)
   ELSE IF (RA IN OPSET1) AND (ICON IN OPSET2) THEN
      BEGIN
      EMIT(219);
      EMIT(VAL2)
      END
   ELSE ERROR(4);

(* RST INSTRUCTION *)

22:BEGIN
   IF (VAL1 MOD 8 <> 0) OR (VAL1 > 56) THEN ERROR(4);
   EMIT(199 + VAL1)
   END;

(* LD INSTRUCTION *)

23:IF R8 IN OPSET1 THEN
      IF R8 IN OPSET2 THEN EMIT(64+VAL2+VAL1*8)
      ELSE IF CON IN OPSET2 THEN
         BEGIN
         EMIT(6 + VAL1*8);
         EMIT(VAL2)
         END
      ELSE IF (RA IN OPSET1) AND (ICON IN OPSET2) THEN
         BEGIN
         EMIT(58);
         EMITW(VAL2,ADM2)
         END
      ELSE IF 'IX,IY!*OPSET2<>'! THEN EMITXY(70 + VAL1*8)
      ELSE IF (RA IN OPSET1) AND (IBD IN OPSET2) THEN
         EMIT(10 + VAL2*16)
      ELSE IF RIR IN OPSET2 THEN EMITED(87+VAL2)
```

```
   ELSE ERROR(4)
ELSE IF ('RBD,RX,RY,RH,RSP!*OPSET1<>'!) AND (CON IN OPSET2)
   THEN BEGIN
   EMITXY0(1 + VAL1*16);
   EMITW(VAL2,ADM2)
   END
ELSE IF ICON IN OPSET2 THEN
   IF RA IN OPSET1 THEN
      BEGIN
      EMIT(58);
      EMITW(VAL2,ADM2)
      END
   ELSE IF 'RH,RX,RY!*OPSET1<>'! THEN
      BEGIN
      EMITXY0(42);
      EMITW(VAL2,ADM2)
      END
   ELSE IF 'RBD,RSP!*OPSET1<>'! THEN
      BEGIN
      EMITED(75 + VAL1*16);
      EMITW(VAL2,ADM2)
      END
   ELSE ERROR(4)
ELSE IF ICON IN OPSET1 THEN
   IF 'RH,RX,RY!*OPSET2<>'! THEN
      BEGIN
      EMITXY(34);
      EMITW(VAL1,ADM1)
      END
   ELSE IF 'RBD,RSP!*OPSET2<>'! THEN
      BEGIN
      EMITED(67+VAL2*16);
      EMITW(VAL1,ADM1)
      END
   ELSE IF RA IN OPSET2 THEN
      BEGIN
      EMIT(50);
      EMITW(VAL1,ADM1)
      END
   ELSE ERROR(4)
ELSE IF 'IX,IY!*OPSET1<>'! THEN
   IF R8 IN OPSET2 THEN EMITXY(112 + VAL2)
   ELSE IF CON IN OPSET2 THEN
      BEGIN
      EMITXY(54);
      EMIT(VAL2)
      END
   ELSE ERROR(4)
ELSE IF (IBD IN OPSET1) AND (RA IN OPSET2) THEN
   EMIT(2 + VAL1*16)
ELSE IF (RSP IN OPSET1) AND ('RH,RX,RY!*OPSET2<>'!) THEN
   EMITXY0(249)
ELSE IF (RIR IN OPSET1) AND (RA IN OPSET2) THEN
   EMITED(71 + VAL1)
ELSE ERROR(4);
```

```
(* EQU PSEUDO-OP *)

24:BEGIN
   IF PDFOUND<>0 THEN
      IF (PDFOUND IN '1..14!) OR (PDFOUND IN '30..31!) THEN
         INSST(PSYM,PDFOUND,TRUE,ADM1,FALSE)
      ELSE ERROR(12)
   ELSE INSST(PSYM,VAL1,FALSE,ADM1,FALSE);
   I := PC;
   PC := VAL1;
   ADDRESS;
   PC := I
   END;

(* ORG PSEUDO-OP *)

25:BEGIN
   IF PSYM<>'          ' THEN
      INSST(PSYM,VAL1,FALSE,ABSOLUTE,FALSE);
   IF ASMTYPE = RELATIVE THEN RELSAVE:= PC;
   IF OBJCNT<>0 THEN
      BEGIN
      WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
      OBJCNT := 0
      END;
   PC := VAL1;
   ASMTYPE := ABSOLUTE
   END;

(* REL PSEUDO-OP *)

26:BEGIN
   IF ASMTYPE = ABSOLUTE THEN PC := RELSAVE;
   ASMTYPE := RELATIVE;
   IF PSYM<>'          ' THEN INSST(PSYM,PC,FALSE,EXTDEFR,FALSE);
   IF OBJCNT<>0 THEN
      BEGIN
      WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
      OBJCNT := 0
      END;
   END;

(* END PSEUDO-OP *)

27:BEGIN
   STOP := TRUE;
   NOTEND:= FALSE;
   FOR I:=1 TO 6 DO HEADER'I! := '          ';
   END;

(* DB PSEUDO-OP *)

28:REPEAT
   SKPBLNK;
```

```
   IF INP^.IPOS!=',' THEN IPOS := IPOS + 1;
   IF INP^.IPOS!=SQUOTE THEN
      BEGIN
      I := STRING;
      FOR J := 1 TO I DO EMIT(ASC^.ORD(STR^.J!)!)
      END
   ELSE BEGIN
      GETOP(OPSET1,VAL1,ADM1);
      IF ADM1 IN ^ABSOLUTE,IMMEDIATE! THEN EMIT(VAL1)
      ELSE ERROR(4)
      END
   UNTIL (IPOS>=IWIDTH) OR (INP^.IPOS!<>',');

(* DW PSEUDO-OP *)

29:REPEAT
   GETOP(OPSET1,VAL1,ADM1);
   EMITW(VAL1,ADM1)
   UNTIL (IPOS>=IWIDTH) OR (INP^.IPOS!<>',');

(* DS PSEUDO-OP *)

30:IF CON IN OPSET1 THEN
      BEGIN
      IF OBJCNT<>0 THEN
         WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
      OBJCNT := 0;
      PC := PC + VAL1
      END
   ELSE ERROR(4);

(* EXT PSEUDO-OP *)

31:IF SRCHST(PSYM,STHEAD) THEN
      BEGIN
      IF PASS1 THEN WITH STFOUND^ DO
         IF ADMODE=RELATIVE THEN ADMODE := EXTDEFR
         ELSE ADMODE := EXTDEFA
      END
   ELSE INSST(PSYM,0,FALSE,EXTREF,FALSE);

(* TITLE PSEUDO-OP *)

32:BEGIN
   WHILE(IPOS<IWIDTH) AND (INP^.IPOS!<>SQUOTE) DO
      IPOS := IPOS + 1;
   J := STRING;
   IF J>0 THEN
      BEGIN
      FOR I := J+1 TO IWIDTH DO STR^.I! := ' ';
      FOR I := 1 TO 6 DO
         PACK(STR,(I-1)*10+1,TITLE^.I!)
      END
   END;
```

```
(* MACRO PSEUDO-OP *)

33:;

(* MEND PSEUDO-OP *)

34:;

(* GEN PSUEDO-OP *)
35:PRGEN := TRUE;

(* NOGEN PSEUDO-OP *)
36:PRGEN := FALSE;

(* LIST PSEUDO-OP *)
37:LIST  := TRUE;

(* NOLIST PSEUDO-OP *)
38:LIST  := FALSE;

(* XREF PSEUDO-OP *)
39:NOXREF:= FALSE;

(* PAGE PSEUDO-OP *)
40:BEGIN
   NOSUPRES := FALSE;
   LINECNT := LMAX+10
   END;

(* ONLY8080 PSEUDO-OP *)
41:ONLY8080 := TRUE;

(* HEADER PSEUDO-OP *)
42:IF PASS2 THEN BEGIN
   NOSUPRES := FALSE;
   WHILE(IPOS<IWIDTH) AND (INP^[IPOS]<>SQUOTE) DO
      IPOS := IPOS + 1;
   J := STRING;
   IF J>0 THEN
      BEGIN
         FOR I := J+1 TO IWIDTH DO STR^[I] := ' ';
         FOR I := 1 TO 6 DO PACK(STR,(I-1)*10+1,HEADER^[I])
      END
   END;

(* MESSAGE PSEUDO-OP *)
43:IF PASS2 THEN BEGIN
   WHILE (IPOS<IWIDTH) AND (INP^[IPOS]<>SQUOTE) DO
      IPOS := IPOS + 1;
   J := STRING;
WRITE(' ');
   IF J>0 THEN
      FOR I:=1 TO J DO WRITE(STR^[I]);
   WRITELN;
END;
```

```
                (* DISPLAY PSEUDO-OP *)
                44:IF PASS2 THEN BEGIN
                    I := PC; (* SAVE PC *)
                    PC := VAL1; (* VALUE TO BE DISPLAYED *)
                    ADDRESS; (* DISPLAY VALUE *)
                    PC := I; (* RESTORE PC *)
                   WRITE('          ');
                   WRITE(HEX(VAL1 DIV 4096),HEX(VAL1 DIV 256),HEX(VAL1 DIV 16),HEX(VAL1));
                   WRITELN(' HEXADECIMAL');
                END;

                (* DUMMY1 *)
                45:;

                (* DUMMY2 *)
                46:;

                (* DUMMY3 *)
                47:;

                (* DUMMY4 *)
                48:;

              END
               END
            END; (* END LINE PROCESSING *)
         IF (PASS2 AND LIST) AND NOSUPRES THEN
            BEGIN
            IF LINECNT > LMAX THEN ZPAGE;
            LINECNT := LINECNT + 1;
            J := OWIDTH;
            WHILE (J>1) AND (OLINE^[J]=' ') DO J := J-1;
            FOR I := 1 TO J DO WRITE(LST,OLINE^[I]);
            WRITELN(LST)
            END
         UNTIL STOP;
      IF PASS2 AND NOTEND THEN ERROR(2);
      IF PASS2 AND (ERRCNT<>0) THEN
         BEGIN
         WRITELN(LST,ERRCNT:5,' PROGRAM ERRORS');
         WRITELN(ERRCNT:5,' PROGRAM ERRORS')
         END
      ELSE IF PASS2 AND (ERRCNT=0) THEN
         BEGIN
         WRITELN(LST,' NO PROGRAM ERRORS');
         WRITELN(' NO PROGRAM ERRORS')
         END
   END;

(*****************************************************************)

BEGIN
   STRTIME := CLOCK;
   INIT;
```

```
    PASS(1);
    REWRITE(OBJ);
    OBJCNT := 0;
    PASS2 := TRUE;
    OUTESD(STHEAD);
    IF OBJCNT<>0 THEN
       WRITELN(OBJ,'&',HEX(CKSUM DIV 16),HEX(CKSUM));
    OBJCNT := 0;
    CKSUM  := 0;
    PASS(2);
    IF OBJCNT<>0 THEN WRITELN(OBJ,'&',HEX(CKSUM DIV 16),
                      HEX(CKSUM));
    WRITELN(OBJ,'$');
    LINECNT := 100;
    SPOS := 0;
    IF SYMTAB THEN DUMPST(STHEAD);
    WRITELN(LST);
    WRITELN(LST,CLOCK-STRTIME:7,' MILLISECONDS USED IN THIS ASSEMBLY');
    WRITELN(LST,' THIS ASSEMBLY MADE ON ',VDATE,' AT ',VTIME);
END.
END.
```